

ON THE FRONT LINE OF GAME INNOVATION

Game DEVELOPER

FEBRUARY 1999

Insights from Warren Spector:

RPGs FOR THE NEW MILLENNIUM

PAGE
19

Advanced
Bounding Boxes

PAGE
46

Implementing a
Coordinated Movement
System

PAGE
60

Postmortem:
Shiny's Wild 9



U.S. \$5.95 CANADA \$6.95



0 73361 87387 3 02>
DISPLAY UNTIL FEBRUARY 25, 1999

Any workstation
can do graphics. Only one
is specifically designed to
make you say,

It's

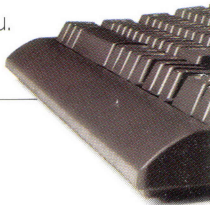
Introducing the newest addition to the Silicon Graphics® family of visual workstations. The operating system is Windows NT®. And the computer is pure Silicon Graphics. Featuring our Integrated Visual Computing (IVC) architecture with the Cobalt™ graphics chipset, these breakthrough machines move graphics data six times faster than AGP 2X-based workstations*—a massive 3.2GB per second. What's more, our IVC architecture integrates 10/100 Ethernet, IEEE-1394†, USB and professional video I/O for a blistering 12-fold increase in bandwidth over other PC workstations.** Finally, you can get the outrageous 2D, 3D and digital media performance you expect from Silicon Graphics—for about the price of a mere clone. The Silicon Graphics 320™ workstation starts at just \$3,395. Or choose the Silicon Graphics 540™ workstation†† from only \$5,995. Supporting up to four Intel® Pentium® II Xeon™ processors, it's the most scalable Windows NT workstation around. Either way, your computer will scream. And so will you.

STARTING
@
\$3,395

[Silicon Graphics 320 Visual Workstation]



- Silicon Graphics Integrated Visual Computing architecture with Cobalt graphics chipset
- Supports up to two Intel® Pentium® II processors (up to 450MHz)
- Ultra ATA or optional Ultra2 SCSI drives up to 28GB total capacity
- Microsoft® Windows NT® 4.0
- Scalable up to 1GB ECC SDRAM
- Add the SuperWide™ 17.3" Silicon Graphics 1600SW™ digital flat panel monitor (shown) for only \$2,495



alive



SiliconGraphics

To get information, find a local reseller or to order, call | 888 SGI-7509 or visit us at www.sgi.com/go/visual



CREATIVE

WWW.SOUNDBLASTER.COM



It's a dark cave. Inside your head, you hear your heart pounding.
Damp **echoes** resonate from the slow **drip** of water. A low **rumble**.

Click — a gun is being cocked to your right.

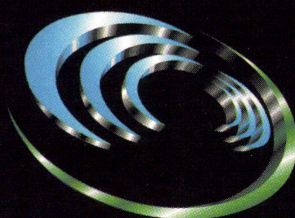
You spin. You duck. You freeze in fear.

Bullets tear past you and **ricochet** from every angle.

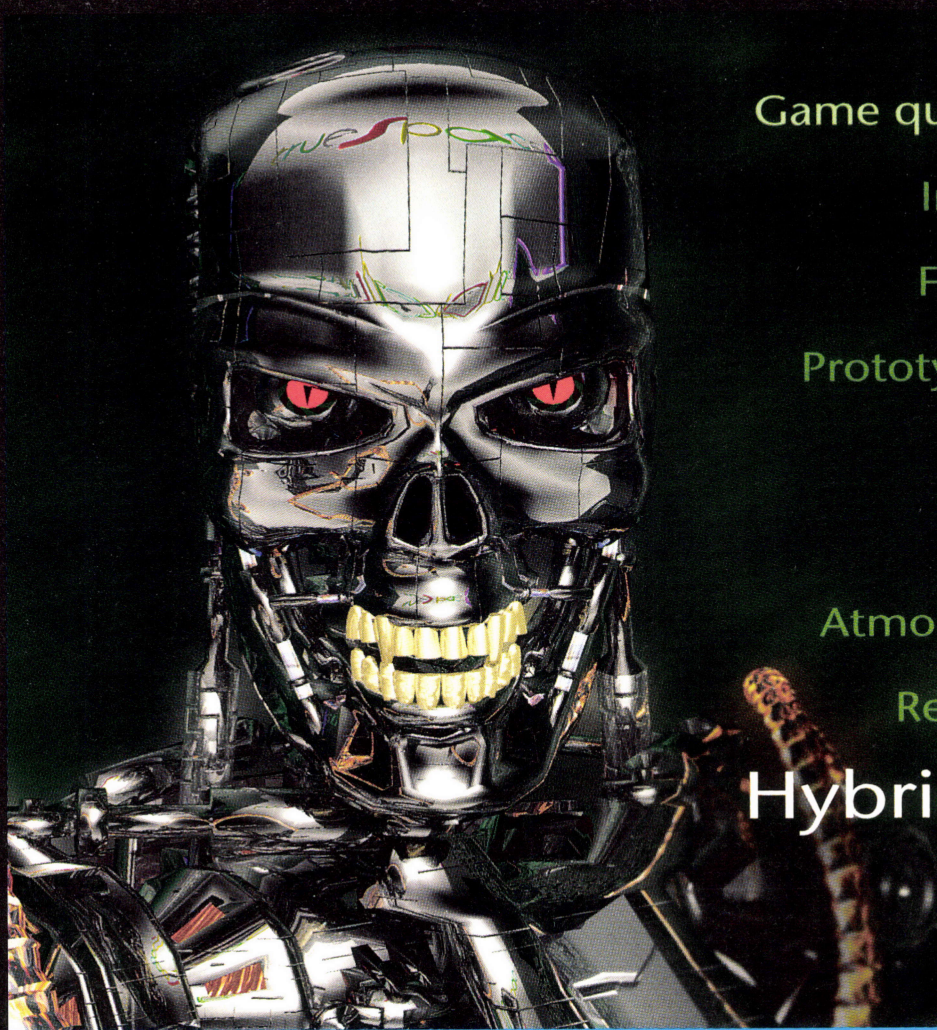
You can't see her, but you know exactly where she is.

Great games deserve great audio. Look for this logo or visit
www.env-audio.com/listen/

So real it has to be live!



**Environmental
Audio**TM
by **CREATIVE**[®]



True 3D interface
Game quality acceleration

Inverse kinematics

Forward dynamics

Prototyping with scripts

Precise collision

Nurbs • Bones

Atmospheric rendering

Reflectance shaders

Hybrid radiosity

\$595

Somebody's gonna be pissed!

trueSpace 4

BORN TO ACCELERATE

See the difference for yourself

1.800.351.7620 www.caligari.com/gd

Caligari Corporation

1959 Landings Drive, Mountain View, CA 94043 • tel: 650.390.9600 • fax: 650.390.9755

Copyright © 1998 Caligari Corporation. All rights reserved. Caligari & trueSpace are trademarks of Caligari Corporation. Cybernetic Assassin by Paul Saff.



FEBRUARY 1999

FEATURES

36 Remodeling RPGs for the New Millennium

Today's computer role-playing games owe their origins to the paper- and die-based games of yore. But isn't it time developers quit rebuilding last year's RPG hit? Spector believes that this genre is only limited by the imagination, and offers several design tools for creating a new role-playing experience.

BY WARREN SPECTOR

48 Implementing Coordinated Movement

Pottinger builds upon the issues he laid out last month in this practically oriented lesson on intelligent movement algorithms for groups of game units.

BY DAVE C. POTTINGER

60 Postmortem: Shiny Entertainment's WILD 9

Can you imagine inheriting a development project from a now defunct team? Which assets would you keep? How do you begin to unravel someone else's undocumented physics engine? Shiny's WILD 9 team answered the challenge, churning out this critically acclaimed PlayStation title in record time.

BY DIDIER MALENFANT

COLUMNS

19 Graphic Content

When Two Hearts Collide

BY JEFF LANDER

27 Artist's View

Playing God: Creating Environments in RT3D

BY MEL GUYMON

34 Hard Targets

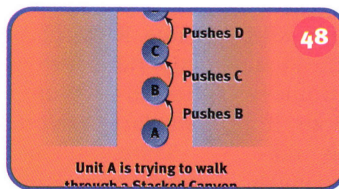
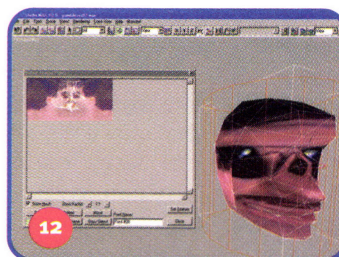
3Dfx Against the Rest

BY OMID RAHMAT

72 Soapbox

Garage Developers are Not Dead

BY LUKE AHEARN



DEPARTMENTS

6 Game Plan

BY ALEX DUNNE

9 Bit Blasts

This month's new product announcements, game industry news, and a review of Sven Technologies' Surface Suite Pro.

70 Creative Careers

What are the best jobs in the field?

COVER: "Wex Major, Boomer McTwist, and B'Angus" from the PlayStation game *WILD 9* by Shiny Entertainment (published by Interplay Productions). Characters designed by Kevin Munroe. Cover image designed by Kevin Munroe with colors by Klaus Lynged.

Landing the da Vinci Deal

Last fall, I had the fortune to find myself in a little town in France called Amboise, located about two hours outside of Paris by train. While wandering around town one day, I came upon an impressive estate which I quickly learned was Leonardo da Vinci's final residence. Da Vinci was given this residence (called Chateau du Clos-Luce, or "house of light") by King Francois I in 1516. In return, the king wanted da Vinci simply to continue his artistic and scientific explorations. If some invention suited Francois' military objectives (as some did), so much the better for the king. Essentially, da Vinci was given free creative reign to do what he was already doing, and in return his patron enjoyed the fruits of da Vinci's ideas.

While at Clos-Luce, I couldn't help but think about an analogy to this scenario within our own industry. Like Francois, today's consumer hardware manufacturers rely on the technical and creative talents of game developers. To a large extent, games are the *raison d'être* for these hardware manufacturers. That's why these hardware companies throw you parties at trade shows, send you free hardware, help you market your game, get you retail shelf space, offer bundling deals, and on occasion a hardware manufacturer will simply shower you with some cash. In short, they need your talents. But while incentives can be difficult to refuse, especially if you're a small developer in need of all of the above, Rob Wyatt of Dreamworks Interactive suggests that you ask the following questions before you sign on the dotted line.

The first question you should pose is to yourself: "Why me?" Why do you think this hardware manufacturer is so hot to get your support of its upcoming product? Sure, *you* know your game will be incredible, but does your prospective patron have the same vision for it as you? Understand what that company expects from your game before you agree to implement any changes in it. If there are differing expectations for the game, you'll have problems down the line.

Second, decide early whether the features or performance gained warrant the extra work. If what's being asked of you won't make your game better, the deal may be skewed substantially in their favor. You're lengthening your development cycle and/or cutting out the development of important features in order to implement irrelevant (or worse yet, harmful) "features" to the game. Remember that by far your most important customer is the player, and that all decisions should be made with that fact in mind.

Third, find out how exactly your patron hardware manufacturer will support your efforts. Will they actually help code certain sections of the game? Find out how much documentation and developer support will be provided before you agree to anything. If you hit a roadblock while trying to implement something for this company, you should be able to get answers to technical questions quickly and easily.

Inquire about the exclusivity of your deal. If you agree to support the Brand X API, does that preclude you from working with another hardware company? Knowing what you cannot do is as important as knowing what you must do.

If you are planning to reuse or license your game engine to others some day, how does an agreement with a hardware company affect those strategies? Find out what lasting legacies you'll have to live with after the current game ships.

Finally, in the event that you've just finished this column and thought to yourself, "I wish I had the problem of too many suitors — where's *my* patron?", take heart. Remember that the universe of game developers is far larger than the combined developer relations staffs at all of the consumer hardware companies out there. Don't wait for them to find you. After all, it was da Vinci who first introduced himself to Francois and offered his services, not the other way around. ■

Alex Dunne

ON THE FRONT LINE OF GAME INNOVATION Game DEVELOPER

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.2228 w: www.gdmag.com

Publisher

Cynthia A. Blair cblair@mfi.com

EDITORIAL

Editorial Director

Alex Dunne adunne@sirius.com

Managing Editor

Tor D. Berg tdberg@sirius.com

Departments Editor

Wesley Hall whall@sirius.com

Art Director

Laura Pool lpool@mfi.com

Editor-At-Large

Chris Hecker hecker@d6.com

Contributing Editors

Jeff Lander jeffl@darwin3d.com

Mel Guymon mel@surreal.com

Omid Rahmat omid@compuserve.com

Advisory Board

Hal Barwood LucasArts

Noah Falstein The Inspiracy

Brian Hook id Software

Susan Lee-Merrow Lucas Learning

Mark Miller Harmonix

Paul Steed id Software

Dan Teven Teven Consulting

Rob Wyatt DreamWorks Interactive

ADVERTISING SALES

Western Regional Sales Manager

Alicia Langer alanger@mfi.com t: 415.905.2156

Eastern Regional Sales Manager/Recruitment

Ayrien Houchin ahouchin@mfi.com t: 415.905.2788

ADVERTISING PRODUCTION

Vice President Production Andrew A. Mickus

Advertising Production Coordinator Dave Perrotti

Reprints Stella Valdez t: 916.983.6971

MILLER FREEMAN GAME GROUP MARKETING

Group Marketing Manager Gabe Zichermann

MarComm Manager Susan McDonald

Marketing Coordinator Izora Garcia de Lillard

CIRCULATION

Vice President Circulation Jerry M. Okabe

Assistant Circulation Director Mike Poplaro

Circulation Manager Stephanie Blake

Circulation Assistant Kausha Jackson-Crain

Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.

President Libby Abramson

720 Post Road, Scarsdale, New York 10583

t: 914.723.4700 f: 914.723.4722

e: abramson@prodigy.com

Miller Freeman

A United News & Media publication

Chairman-Miller Freeman Inc. Marshall W. Freeman

President/COO Donald A. Pazour

Senior Vice President/CFO Warren "Andy" Ambrose

Senior Vice Presidents H. Ted Bahr, Darrell Denny

Galen A. Poss, Wini D. Ragus, Regina Starr

Ridley, Andrew A. Mickus, Jerry M. Okabe

Vice President/SD Show Group KoAnn Vikoren

Senior Vice President/Systems and Software

Division Regina Ridley

BPA International Membership

Applied for March 1998



© 1998 Paramount Pictures

Creativity at Warp Speed

Inspiration descends in flashes, and when it does, speed is the key to transforming that inspiration into impressive animation and special effects. When Star Trek® Starfleet Academy™ game developer Interplay Productions needed a 3D graphics accelerator that would enable them to produce the look and feel of the Star Trek feature films and television shows, they turned to Evans & Sutherland's Accel boards and REALimage® technology.

Our newest board, the AccelGALAXY, uses E&S REALimage 2000 technology and features 20MB of 3DRAM frame-buffer memory and 32M of CDRAM texture memory. Just one board can handle dual screens and support 3D stereo glasses. In addition, the AccelGALAXY offers 4M Z-buffered triangles per second, 90M textured pixels per second, and high quality anti-aliasing.



Interplay Productions

Animator Michael Sherak

is the creative force behind the Starfleet Academy game.

"Ever since I've had my Accel board, I've been impressed by it. Every day I load up my 3D program and move things around and animate and texture-mount and build models. If I didn't have the card . . ., I probably would still be halfway back on Starfleet, trying to complete it."

For more information, call us at 800-444-5699, or visit our website at www.es.com.



EVANS & SUTHERLAND

cakewalk

Pro Audio™

24-bit
96 kHz

Vector Automation

Real-time
MIDI Plug-ins

Sync AVI MPEG
QuickTime

Now Shipping

The new version of the leading MIDI and
digital audio software

www.cakewalk.com
888-CAKEWALK



All trademarks mentioned are held by their respective owners.

BIT

Blasts

News from the World of Game Development



New Products: Kaydara's Filmbox, Newtek's Frame Factory, and Credo's PowerModels **p. 9**



Industry Watch: Game Grammys, Senators' ratings, and medical breakthroughs **p. 10**



Product Reviews: Paul Steed reviews Surface Suite Pro **pp.12-14**



New Products

by Wesley Hall

Motion Capture Made Easy

KAYDARA is shipping Filmbox 1.5, its real-time production software.

A suite of motion capture, animation, and interactive rendering tools, Filmbox is designed to simplify motion capture mapping and editing. The suite integrates into the production workflow for game, television, and film productions, and supports all major 3D packages. Feature highlights in Filmbox 1.5 include: HumanIK, Motion Sculpting, and Open Reality. HumanIK is a new tool for automatically attaching motion capture data to skeletons. It connects capture data and character joints for you. Motion Sculpting's new control curves allow you to apply offsets to captured animation, and give you a secondary level of animation control. Open

Reality is a cross-platform C++ developer kit designed specifically for the creation of real-time plug-ins for Filmbox. Developers have complete access to Kaydara's performance architecture, as well as import/export access to all major 3D products such as Alias|Wavefront Power Animator/Maya, Kinetix 3D Studio Max, Newtek Lightwave, and Softimage 3D.

Available on both the Windows NT (Intel and Alpha) and Silicon Graphics IRIX operating systems, Filmbox 1.5 is immediately available with pricing starting at \$4,995. The Open Reality SDK is also available separately, directly from Kaydara.

■ **Kaydara Inc.**
Montreal, Quebec, Canada
(888) 842-6842
<http://www.kaydara.com>

Affordable Frame Factory

NEWTEK recently launched and shipped Frame Factory, its first professional hardware and software digital video, paint, and animation solution.

Frame Factory is an affordable hardware/software integrated solution (it sells for under \$4,000) that is specifically designed for 3D animation, real-time video capture and playback, 2D animation, video paint, 2D and 3D video manipulation, and video processing. It gives you complete control over modeling, texturing, painting, animating, rendering, rotoscoping, compositing, and

final output from one desktop system. The package includes an uncompressed ITU-R-601 video I/O board, Lightwave 3D 5.6, and Aura, Newtek's video paint production software tool — all optimized to work together on Windows NT.

Newtek's Frame Factory for Intel is immediately available for a suggested retail price of \$3,995. Minimum system requirements are Windows NT with service pack 3, a Pentium 166MHz, 96MB RAM, drive system capable of 21MB/Sec sustained data rate, and available bus mastering PCI slots for Frame Factory board and SCSI controller.

■ **Newtek**
San Antonio, Tex.
(210) 370-8000
<http://www.newtek.com>

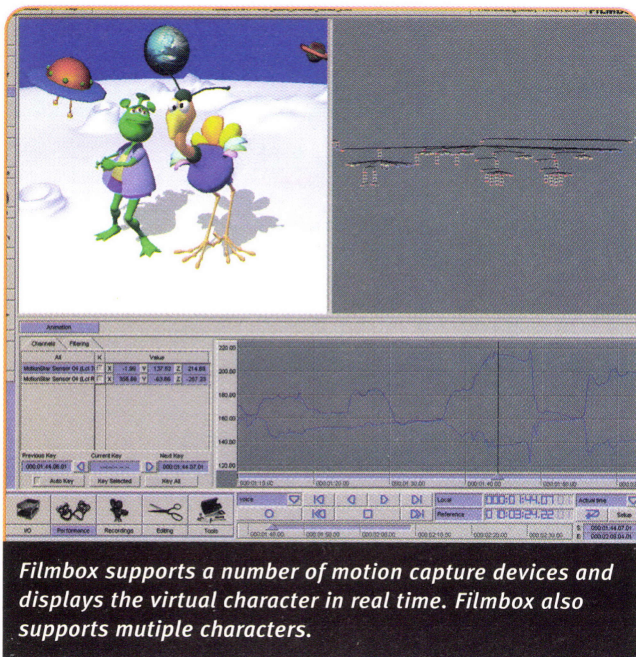
Ready-Made Models

CREDO INTERACTIVE has released Powermodels, a collection of ready-to-animate 3D character models.

The Powermodels collection includes 24 multiracial human characters from Zygote Media Group, 13 human characters from Geo-Metricks, and six original insects created by Credo Interactive. With Powermodels, Life Forms users (as well as users of Max and Lightwave) can now populate their projects with diverse characters, create multiple character interaction, and quickly create large crowd scenes — even if the crowd is a swarm of Pixar-inspired bugs and ants.

Powermodels is immediately available and sells for a suggested retail price of \$249. The models are compatible with Life Forms, 3D Studio Max, and Lightwave 3D formats.

■ **Credo Interactive**
Vancouver, B.C., Canada
(604) 291-6717
<http://www.credo-interactive.com>



Filmbox supports a number of motion capture devices and displays the virtual character in real time. Filmbox also supports multiple characters.



Industry Watch

by Alex Dunne

GAME GRAMMYS? For all of you composers out there who complain that you don't get enough credit for your work, maybe a Grammy Award will ease your suffering. It was recently revealed that the National Academy of Recording Arts and Sciences met with about a dozen of the game industry's top music professionals (including George Alistair Sanger, Bobby Prince, Mark Miller, Rob Hubbard, and Tommy Tallarico) in San Francisco, Calif., to discuss the feasibility of establishing a Grammy Award for our industry. Think of something along the lines of "Best Original Soundtrack from an Interactive Game." You can thank Chance Thomas, a composer at Yosemite Entertainment, for getting the ball rolling with the Grammys.

MGM INTERACTIVE AND ELECTRONIC ARTS entered into a worldwide development and distribution agreement for MGM's next slate of interactive titles. In the deal, MGM's upcoming titles, including *TOMORROW NEVER DIES* and *ROLLERBALL*, will be sold, marketed and distributed by EA worldwide.

SEGA SOLD 150,000 DREAMCASTS on its first day of sales in Japan, and the company is shooting to sell one million units by the end of March.

SOFTIMAGE AND NINTENDO announced a partnership to design and create game development tools for the N64 home video game console. Under the development agreement, Nintendo and Softimage will collaborate on defining NIFF (Nintendo Intermediate File Format) 2.0 and Softimage is adding new features to Softimage 3D 3.8 to support N64 development. The Nintendo NIFF 2.0 development environment and the Softimage tools will ship concurrently to all authorized Nintendo developers in January 1999.

PSYGNOSIS AND CLOTHING FIRM DIESEL announced that they will co-promote the upcoming title, *G-POLICE WEAPONS OF JUSTICE*. Diesel has designed jackets, pants, T-shirts and sweatshirts based on the futuristic game world, for

sale in the U.S. and Europe. A looping *G-POLICE* video, point-of-sale material, and themed areas will be present in all Diesel stores. This is the second time that the two companies have worked together to promote a game (they promoted the original *G-POLICE* together).

SENATORS Joseph Lieberman (D-Conn.) and Herb Kohl (D-Wis.) released the fourth annual "Video Game Report Card" on the state of game morality, which was conducted by the Minneapolis, Minn.-based National Institute on Media and the Family. Perhaps not surprisingly, the industry took it on the chin once again. Kohl and Lieberman simultaneously praised the industry for adhering to the five-year-old ESRB voluntary ratings system and blasted the gore found in its most violent games. In citing the worst offenders on the market this past Christmas, the report gave *DUKE NUKEM: TIME TO KILL* the lowest "KidScore" rating, followed closely by *BIO FREAKS* and *MORTAL KOMBAT 4*. The report, and ratings for individual games, can be found at <http://www.mediaandthefamily.com>.

GT INTERACTIVE signed affiliate label agreements with Sega PC and Sega Soft Networks, whereby it will handle the North American sales and distribution of PC titles from both Sega companies. The agreements cover upcoming Sega games including *ENEMY ZERO*, *VIGILANCE*, and *SEGA RALLY 2 CHAMPIONSHIP*.

DOCTORS AT THE ROYAL FREE HOSPITAL in London announced the results of a test they conducted to treat

patients with stress-induced irritable bowel syndrome (which you'd be surprised to know affects almost 10 percent of the population). The patients were treated using a game developed by a company called *ULTRAMIND*, which relied on sensitive sensors to monitor stress levels and provide real-time biofeedback. The more relaxed the patient, the better the progress in the game. The software developers hope that their system will help reduce anxiety levels, improve workplace performance, train and educate children to deal with various situations, and treat a variety of psychosomatic disorders. Ah, the wonders of our craft.



DUKE NUKEM: TIME TO KILL earned the lowest "Kidscore" from Senators.

UPCOMING EVENTS CALENDAR

February 9-12, 1999

Milia 99
Palais des Festivals
Cannes, France
Cost: \$655
<http://www.reedmidem.milia.com>

February 17-20, 1999

TED 9
Monterey Convention Center
Monterey, Calif.
Cost: \$2,250
<http://www.ted.com>



Psygnosis and Diesel team up to create game fashion.

NETIMMERSE

Develop your game play, not your game engine.

The Fastest Path from 3D Game Conception to Completion



Screen shot from Darkstorm™
by Mythic Entertainment LLC

NETIMMERSE is a professionally engineered, C++ 3D game engine that allows you to **begin developing content from Day 1**. **Save 6 months of development time** and **years of engineering costs** right out of the gate. Free your software programmers for the **really cool stuff** that game players see on the screen.

Here are some of the reasons why game developers such as **Red Orb Entertainment, GreyStone Technology, MediaStation, Mythic Entertainment, Aeon Entertainment** and others rely on NETIMMERSE:

- **It delivers action** - aggressive culling, portal-based visibility, and hand optimization in critical code
- **It delivers realism** - multi-textures, projected textures, animated textures, environment maps, and a wide variety of lighting effects
- **It delivers advanced animation** - morphing vertices, particle systems, and quaternion-based keyframes
- **It delivers scene management** - dynamic collision detection, continuous level of detail, and 3D picking
- **It's compatible** - provides a plug-in to 3D Studio MAX®, MultiGen's Creator® and Motion Factory's Motivate™, and works with Direct3D, OpenGL and Glide
- **It makes business sense** - licenses include access to full source code, no royalties, and complete support

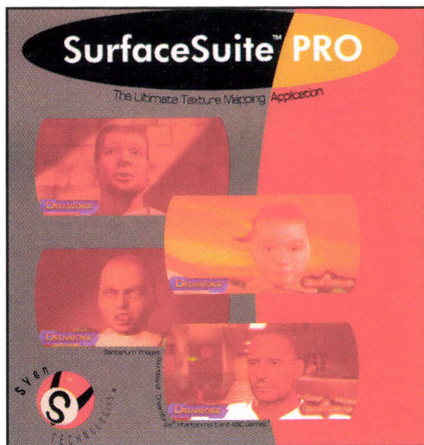
Contact NDL today to get on the 3D game fast track!

Check out The Business
Case for Buying Instead of
Building a 3D Game Engine
on NDL's Web site.

www.ndl.com

sales@ndl.com

Palo Alto, CA Sales Office - tel: 650-328-4388; fax: 650-328-3454
Chapel Hill, NC Headquarters - tel: 919-929-2917; fax: 919-967-3237



SurfaceSuite Pro for Max

by Paul Steed

Over the past six months, I've done quite a bit of research on the manipulation of UV coordinates. My goal is to get the most effective coverage I can on a given mesh for our upcoming title, *QUAKE 3: ARENA*. Because I'm the animation and modeling department at id Software, speed is of the essence for me to do my job effectively.

When creating a character, I typically use a cylindrical mapping projection

and make sure the texture tiles, thus avoiding any unsightly seams. If, however, the UV coverage needs tweaking, I turn to UV Unwrap (a modifier that comes free with Max). Always looking for the next better thing, I got wind of a nice little third-party plug-in for 3D Studio Max called SurfaceSuite Pro by Sven Technologies.

Advertised as the ultimate texture-mapping application, SurfaceSuite Pro for 3D Studio Max consists of four plug-ins: Texturizer, Multimask, Gaussian Blend, and Global Map Generation. Texturizer is the main mapping plug-in and allows you to manipulate UV coordinates. Multimask and Gaussian Blend handle the texture-blending features, and Global Map Generation creates a single composite map from multiple, blended, overlapped textures. Sven also offers all these functions in a stand-alone application called simply SurfaceSuite Pro. But since Texturizer is a plug-in for the modeling and animation package that I use (Max), I decided to give it a closer look.

Installing Texturizer was easy enough. After launching Max and authorizing the plug-in with Sven, I added it to my button set of modifiers. For this exercise, I used a prebuilt mesh based on the head of John Cash, a programmer here at id. Before loading in the head, though, I had to make a material to apply to it. I used a 256x128 texture created in Adobe Photoshop from front and side shots of old Cash. I designed the texture to wrap-around the mesh, joining seamlessly at the back of the head (Figure 1).

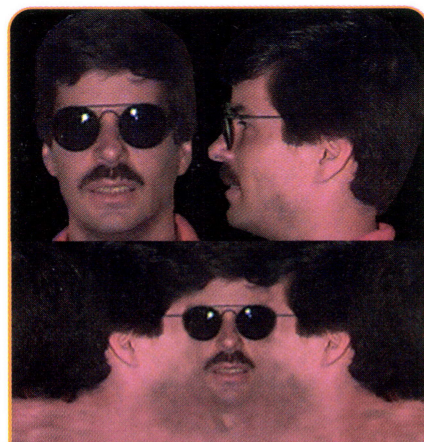


FIGURE 1. Front- and side-views of Cash's head and the resultant seamless texture.

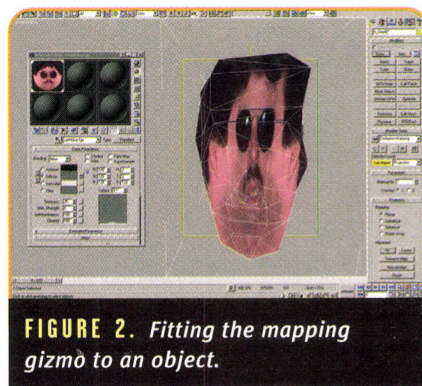


FIGURE 2. Fitting the mapping gizmo to an object.

Next, I loaded up Cash's head model. After assigning the texture in the Modifier menu, I assigned Texturizer to the head. A standard mapping gizmo of the planar variety immediately popped up, oriented from the top. In the manual, Sven recommends using a planar projection instead of a cylindrical or spherical one (although SurfaceSuite supports all three types). Although this is contrary to the technique that I normally use, the manual explains that a planar projection works for 95 percent of the jobs for which you'll use SurfaceSuite. Hmmmm...

I rotated my viewport so that Cash's untextured head appeared straight on. The manual directs you to apply Texturizer in the perspective viewport, but I would have preferred to use a front viewport. I clicked on Subobject and brought up Projection. As planar projection is already selected by default, I clicked on Viewport Align to align the mapping gizmo to the viewport. Finally, selecting Fit expands the gizmo a little to "fit" the object's front-on dimensions (Figure 2).

So far, so good. Clicking Association under the Subobject menu beneath Projection presented me with a menu to create association points. As association points are like making your own UV points on the fly, this sounded neat.

Sven suggests laying down five points on the object's mesh and five corresponding points on the texture that will wrap around it, and then naming them (such as, "Right Eye," "Left Eye," and so on). These five markers, which should correspond to each eye, the nose, and the two corners of the mouth, are used as reference points to apply the texture onto the object properly. Naming these points in that fashion is way too time consuming so, with the Define Texture Points window open, I'd lay a couple points on the texture then a couple points on the mesh. This approach worked fine until I added a point at each ear. Ouch. Unfortunately Texturizer didn't like that too much (Figure 3).

I was pretty sure that the problem was with the planar mapping, so I thought I'd lump myself into that five percent that uses something other than the planar mapping scheme and give cylindrical mapping a poke. I changed the Projection subobject under the Texturizer modifier from Planar to

*Paul Steed has been making low-polygon models and animating since 1992. He's been at id for more than two years now, having worked on *QUAKE 2* and the current project: *QUAKE 3 ARENA*. He very much enjoys being a guy and equally enjoys doing very stereotypical guy things.*

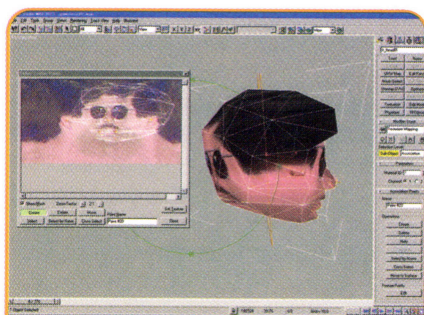


FIGURE 3. *Something's wrong with these association points.*

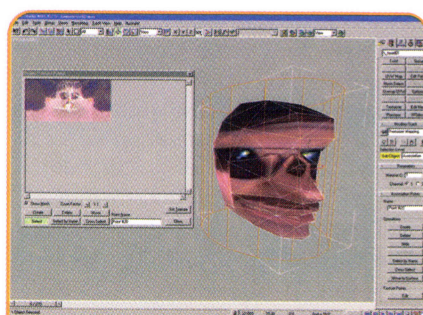


FIGURE 4. *Switching from planar to cylindrical mapping in the middle of the operation isn't advisable.*

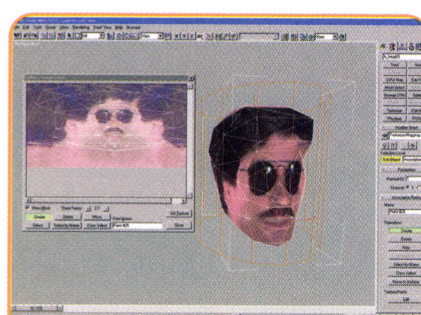


FIGURE 5. *Assigning new association points with a cylindrical mapping type works pretty well.*

Cylindrical, and as you can see, the results weren't pretty (Figure 4). So I deleted the old association points and created new ones using the new cylindrical mapping type. Starting over again, I laid down the prescribed five points in the Define Texture Points window, and did the same on the mesh. That did the trick (Figure 5).

Okay. I have a few small complaints with this process up to now. First, although the texture I was previously editing in the Define Texture Points window was still loaded when I went back to it, Texturizer didn't remember my previous window setting (I previously had a zoom factor setting of 2:1 on the source image). The second quirk I noticed was that, while I could delete editing points on the mesh using my keyboard's [Delete] button, deleting association points in the Define Texture Points window didn't support the [Delete] key. You have to use the Delete button in Texturizer. Finally, when you create a point in the Define Texture Points window, it's given a number value that is consecutive from the number value of the last point created. When a point is deleted, the numbering system still marches on. I guess this is because the random

numbering is simply a temporary name for you to change. It'd be nice if the numbering tied itself directly to the association points.

Adding more association points tightens up the map-to-mesh relationship even more, but I found that knowing where to create these points is an acquired skill. I created some points at each ear, but that caused the mapping to go haywire yet again. I was more successful when I tried tightening up the forehead by creating two points at the front hairline. By creating a point at the back of the head, I was able to remove some weird texture-mapping effects at the side of the mesh. Overall, the cylindrical mapping method worked well, except at the top of the mesh, where cylindrical mapping schemes invariably have problems (Figure 6).

In an attempt to fix problems associated with the placement of the character's hairline, I moved some association points around. It's interesting to see what happens when you move these points: the entire mapping outline of the mesh seen in the window moves around and is updated in the viewport.

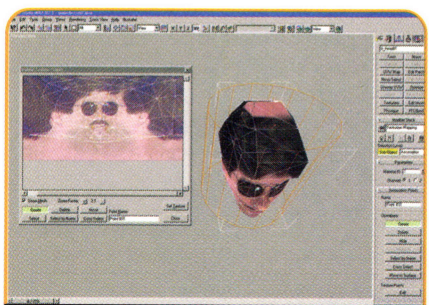


FIGURE 6. *Relative success using the cylindrical mapping method.*

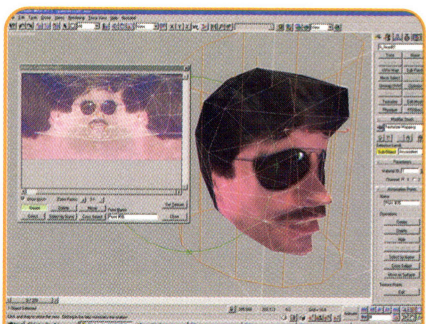


FIGURE 7. *Manipulating association points at the hairline led to problems around the glasses.*

However, I found that attempts to correct the hairline came at a cost: the glasses started looking kind of droopy. So I decided to delete the current association points at the hairline and create new ones at the different spots, but that didn't work. The glasses still drooped (Figure 7).

So I tried adding more association points around the glasses. Here, I got to play with one of the more interesting aspects of Texturizer: the Move to Surface button. When it's on, this feature lets you select and move association points while staying "glued" to the mesh. This deforms and changes the texture as the points are moved around. Unfortunately, it didn't do me much good because I couldn't figure out a comfortable correspondence between moving the association points in the Define Texture Points window and moving points on the surface of the mesh. When do you need to move one and not the other?

I decided to live with the droopy glasses for the time being and concentrate on a problem on the side of the head. In the process of wrapping the texture around the object, an ear disappeared. Because my last attempt to create an association point at the ears turned out to be a failure, I trod carefully, cursor near the Undo button. I created a point by the right ear of our texture subject, but I simply couldn't get the ear to align to the mesh properly.

At this point, I deduced that the default naming of the association points had to do with the dilemma. More specifically, the default names given to the points on the mesh had to correspond to the like-named association points on the texture. And because I'd been adding and deleting

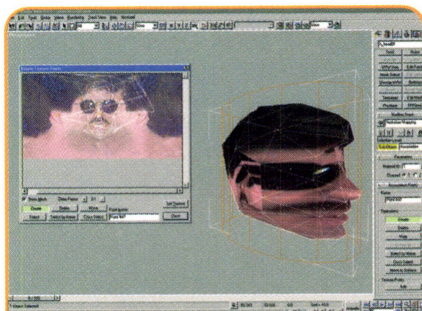


FIGURE 8. *Syncing up association points still didn't fix the ear problem.*

points haphazardly during the course of fitting the texture to the object, default names for corresponding points on the mesh and texture were being assigned different numbers.

To solve the problem, I went to the mesh and created a series of points and then deleted them to resynchronize the default point names. However, even with this problem solved, the results weren't great, as you can see (Figure 8).

Finally, I decided to cut my losses and accept the fact that using Texturizer, poor Cash would have a permanent bad hair day on the right side of his head.

JUDGES SAY... In my opinion, Texturizer is a decent tool, but it has some annoying aspects to it. I found the necessity to name association points in order to get better mapping results to be a bad time sink. This product does feature some powerful multitexture blending capabilities and animatable texture mapping, but I don't find these compelling enough to switch from UV Unwrap. While SurfaceSuite possesses some pretty good functionality, it did nothing to boost my productivity.

Overall, I think SurfaceSuite's Texturizer plug-in for 3D Studio Max is

promising, but ultimately outmatched by UV Unwrap. However, if you're not using Max and don't have a utility of this nature, you should check out Sven's stand-alone version of SurfaceSuite. Softimage, Lightwave, Maya, and Photoshop files can be easily brought into SurfaceSuite Pro, and the package offers the same features as the set of plug-ins — Texturizer, Multimask, Gaussian Blend, and Global Map Generation — without tying you to Max. Texturizer has promise, but for now, my money's on UV Unwrap for UV manipulation in 3D Studio Max. ■

SurfaceSuite Pro for 3D Studio Max: ★★★★★

Company: Sven Technologies
Palo Alto, Calif.

(650) 852-9242

<http://www.sven-tech.com>

Price: \$495 (stand-alone version is \$595)

System Requirements:
Windows 95/98/NT, 3D Studio Max, 32MB RAM and an SVGA video card

Pros

1. Easy to set up.
2. Four plug-ins instead of one.
3. Moving association points around on the mesh is very slick.

Cons

1. Tutorials are a little weak.
2. The editing window doesn't remember settings.
3. Naming infinitely numbered association points slows down the work flow.

3DS MAX? SoftImage? Lightwave? CAD? PolyTrans Translates All Major 3D Formats!

PolyTrans v2
With 3DS MAX® Plugin Support!

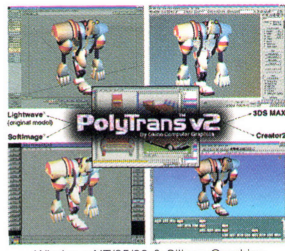


Release 2.2 Now Supports Native 3DS MAX® Plugins, OpenFlight®, Trimmed NURBS, Full IGES 5.3, Unique Animation Conversion (Lightwave®/3DS/DirectX/Renderman), ACIS® 4.2 SAT (solids) and more. Creates "Render Ready" Files!



Download Demos From <http://www.okino.com>

3D Studio® r4 & MAX
Apple® 3DMF®
Alias® triangle
Detailer®
DirectX®
DXF®
IGES 5.3
Imagine®
Lightscape®
Lightwave®
OpenFlight®
OpenGL C Code
POV 2 & 3



Pro/E® .slp
Renderman® RIB
Renderware®
SoftImage® hrc/dsc
SAT 4.2 (ACIS®)
Solid Edge® .Part
STL
Strata® SPro v1.75
trueSpace® 2 & 3
USGS DEM
VistaPro® DEM
VRML™ 1&2
Wavefront® (+NURBS)

"PolyTrans is 'the' premiere 3D conversion tool available today."
—Peter-Joseph Kovach, 3D Book Author & VC++ Dev. Journal Magazine Writer.

"PolyTrans is by far the most versatile (translator) with the most reliable results."
—Joe Morley, Sales & Marketing Director, REM Informatika (3D Models Bank).

"PolyTrans is the only tool we have found that can correctly convert our 3D databases."
—Rav Arell, Architecture Validation Engineer, Intel Corp.

The NuGraf
Rendering System

Or, step up to Okino's **NuGraf** for full material editing, photo-realistic rendering and more!

Okino
Computer Graphics

Toll Free: 1-888-3D-OKINO, WEB: <http://www.okino.com>

T: (905) 672-9328, F: 672-2706, Email: sales@okino.com

© 1998 Okino CG, Toronto, Canada. NuGraf & PolyTrans are registered trademarks, & Okino is a trademark of Okino C.G. All other products mentioned are trademarks of their respective holders. *Apektor © 1998 Newtek, Inc. & VSI.

ON THE FRONT LINE OF GAME INNOVATION

Game DEVELOPER

GAME DESIGNERS

ANIMATORS

GRAPHIC DESIGNERS

MULTIMEDIA PRODUCERS

SOFTWARE DEVELOPERS

Reach these prime prospects with Game Developer mailing list. Please contact Kristen Kohnke at Rubin Response Management Services, Inc. for complete details - 847.619.9800

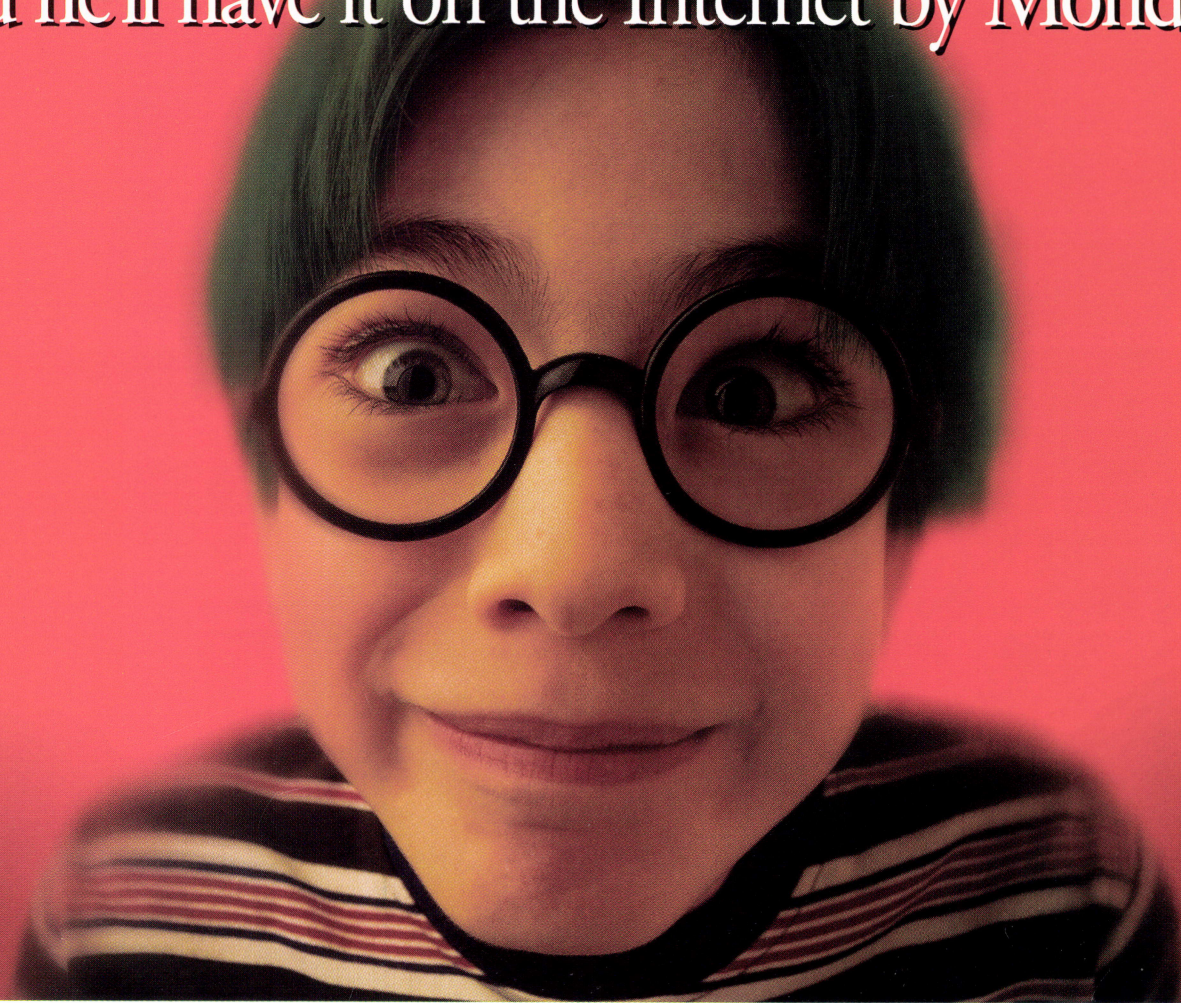


Rubin Response Management Services, Inc.

Technology List Division • (847)619-9800

E-mail kristenk@rubinresponse.com

Release a new game on Friday, and he'll have it on the Internet by Monday.



Unless you use SAFEDISC.

He's just having a little fun. At your expense. In fact he's just one example of how consumer copiers, hackers, and professional pirates cost the PC entertainment, education, and reference CD-ROM publishing industry more than \$3 billion a year. Which is exactly why you need SAFEDISC™.

SAFEDISC prevents CD-Recordable drives and other re-mastering devices from producing useable copies of original CD-ROMs. It does not interfere with game or program playability...and its disc authentication technology ensures that only original discs are playable. SAFEDISC is now enabled at major replicators worldwide. In fact, SAFEDISC-protected titles are being distributed by major publishers throughout North America, Europe and Asia.

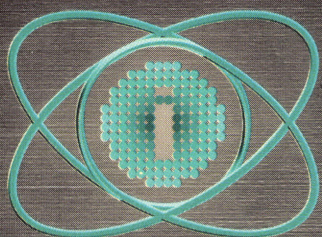
SAFEDISC is from Macrovision Corporation, the leading copy protection technology company in the world. Macrovision offers its security technologies in over 35 countries on six continents. In fact, our video copy protection technology has been the de facto standard for protecting Hollywood's home video movies for over a decade.

Now a truly effective security technology is available to the interactive software market from an industry leader. Call **1-800-900-4229** to find out more. Or visit our website at **www.macrovision.com**. Take the critical step to protect your consumer titles and pre-release software. Make sure you're releasing your titles on the Internet...not him.



Creativity deserves more than

THE MILLER FREEMAN GAME GROUP IS PLEASED TO ANNOUNCE THE



1ST ANNUAL GDC INDEPENDENT GAMES FESTIVAL

Showcasing innovative games from around the world. Encouraging new game ideas.

That's what the **1999 GDC Independent Games Festival** is all about. This festival brings together the best new game ideas in the industry during a three-day showcase, competition, and awards ceremony at the Game Developers Conference in **San Jose (March 15-19, 1999)**. Established to support innovative, new games that lack a publisher, the Independent Games Festival is the ideal place to show off new game concepts, and to see what's coming next. In addition to the festival area at the GDC, four awards will be presented by a panel of game industry experts in these categories:

Grand Prize

Art

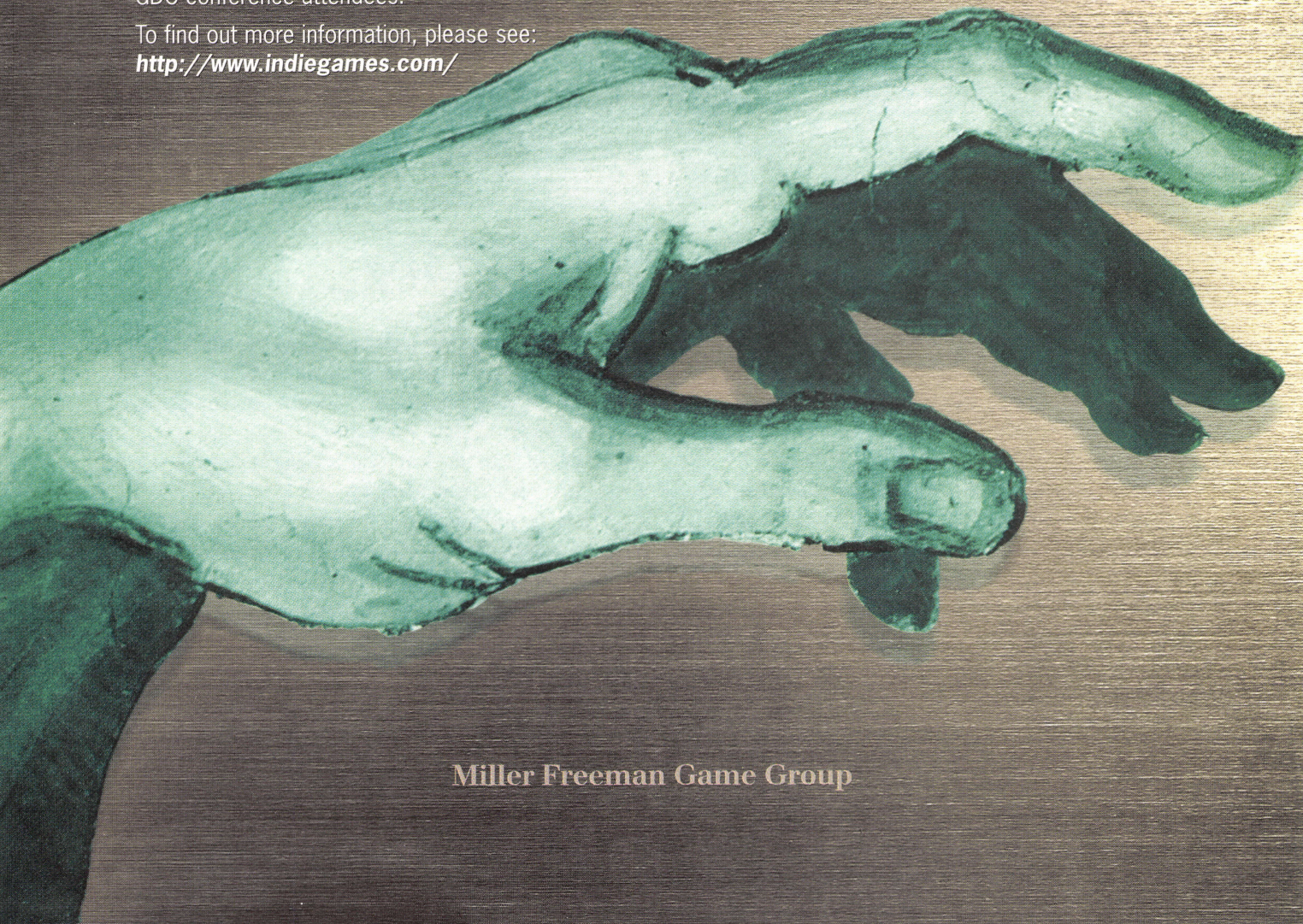
Audio

Game Design

Technical Achievement

A Grand Prize will be given out to the best independent game of the year, and an Audience Award determined by the popular ballot of GDC conference attendees.

To find out more information, please see:
<http://www.indiegames.com/>



Miller Freeman Game Group

just a hand.



Eligibility for Competition

Games produced, financed or initiated by a publisher are not eligible. However, games produced, financed and initiated by an independent developer (defined as any individual or company not currently receiving funding from a publisher), and games that were purchased by a publisher after completion, are eligible.

There are two Festival categories:

Completed Games and Unfinished Games.

◆ Completed games are ready to be played in their entirety, and will only be judged against other completed games. Eligible completed games may not have been released commercially before January 1, 1997, and must be completed by January 1, 1999.

◆ Unfinished games must be playable, and have at least one complete level to demonstrate. Unfinished games will only be judged against other unfinished games.

The Festival will not assume any responsibility for non-disclosure or secrecy violations for its entrants, and the festival may disclose and/or exhibit any or all submissions. The Festival is open to the public, and it should be assumed that all submissions may be seen by members of the public, including game publishers and other game developers.

Games must run on an Intel (or compatible) based PC, a Macintosh, or a Sony Playstation. All submitted games must be made available to the festival jury in diskette, CD-ROM, or DVD format, along with complete printed documentation, and completed official application form.

Selected entries must be shipped prepaid, insured, packed in proper containers and received at the GDC Indie Festival office no later than January 15, 1999.

The jury of the GDC Independent Game Festival will comprise distinguished individuals from the field of game development, serving as judges of the Competition. The jury will judge games and select winners in five categories:

- 1 Grand Prize
- 2 Artistic Award
- 3 Audio Award
- 4 Game Design Award
- 5 Technical Excellence Award

In addition, an Audience Award determined by a popular vote of conference attendees will be given at the Festival.

A Grand Prize award will be given to the individual designated on the official entry form. The games receiving the six awards at the GDC Indie Festival will be profiled in an upcoming issue of Game Developer magazine and on the Gamasutra website.

For further information, official rules, and an official application, check out the GDC Independent Games Festival website at <http://www.indiegames.com>.



- Hardware support for vertex arrays
- 16-bit color or 32-bit true color rendering
- Both fog and fog table support
- Alpha blending, vertex and Z-based fog, video textures, texture lighting
- Single clock bilinear and trilinear texture filtering and texture compositing
- Perspectively correct mip-mapped texture filtering and texture compositing
- Vertex and Z-based reflections, shadows, spotlights, LOD biasing
- Hidden surface removal using 16, 24, or 32-bit Z-buffering
- Dual texel pipe, delivering 2-pixels per clock
- Gouraud and specular shaded polygons
- Integrated DVD/MPEG-2 decode
- Line and edge anti-aliasing
- Single pass multitexturing
- Bump mapping
- Highly optimized 128 bit engine
- 8-bit stencil buffer

WHAT COULD 30 MILLION PEOPLE POSSIBLY SEE IN ATI...

OPTIMIZE FOR ATI AND YOU HAVE 30 MILLION PEOPLE WITH 3D CHIPS READY TO PLAY WITH YOU INSTANTLY



www.atitech.com/gd

© Copyright 1998, ATI Technologies Inc., ATI, and RAGE 128 are trademarks and/or registered trademarks of ATI Technologies Inc. All other company and/or product names are trademarks and/or registered trademarks of their respective manufacturers. Products may not be exactly as shown. Features, performance and specifications may vary depending on operating environment and are subject to change without notice.

When Two Hearts Collide

February is Valentine's month. Spring is in the air. People can meet, fall in love, and have their hearts broken all before their first cup of coffee. I don't think we've reached the point, yet, where we can adequately simulate a broken heart. However, I do think we can reasonably detect whether two people are close enough for their hearts to collide.

Last month, I discussed the use of the dot product and cross product to handle collision detection for 2D applications. This month we'll look at how to apply the same principles to 3D. Most discussions of collision detection for real-time game applications begin with bounding spheres and bounding boxes. These two tests are very rough indicators of whether or not a collision has occurred. Bounding spheres are a very fast method for testing collisions. However, as we saw last month, bounding spheres don't generally provide the best approximation of an object's extents.

Don't Box Me In

An axis-aligned bounding box (AABB) is also a very quick way of determining collisions. The fit is generally better than a bounding sphere (especially if the object you are bounding is a box itself). You can see an AABB on an object in Figure 1. However, once you rotate the object a little, the bounding box may not be nearly as efficient, as you can see in Figure 2.

This discrepancy could clearly lead to cases in which you mistakenly assume that a collision has occurred. But, as a first step, calculating an AABB may not be too bad. We could allow the original bounding box as calculated in Figure 1 to orient along with the object (Figure 3). This is called an oriented bounding box (OBB). It's definitely a better fit than Figure 2, however, I have lost the key benefit of AABBs. Aligned axes make AABBs easy to use in collision detection. Checking

whether a point has entered the box involves only a trivial calculation. With OBBs, checking for collisions is more complicated. In many applications, OBBs may be worth pursuing further, but for a quick first check, I want to stick with AABBs.

So what are the main problems with coding up AABBs? Well, the biggest issue with using AABBs is that they need to be recreated every time the object changes orientation. This means that every vertex must be transformed by the object's matrix and the minimum and maximum extents must be calculated. Listing 1 contains a routine that calculates an AABB for an object.

The noteworthy line in this code is the `MultVectorByMatrix()` call. This function transforms each vertex coordinate of

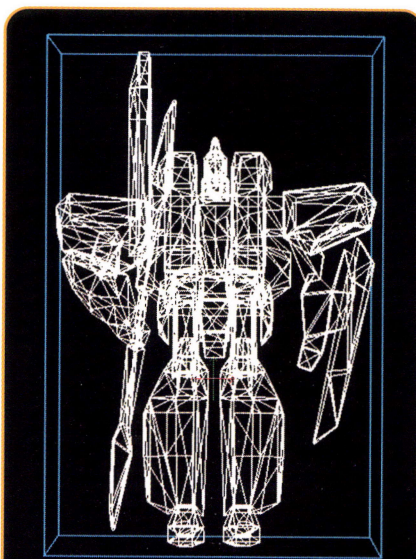


FIGURE 1. Axis-aligned bounding box on an object.

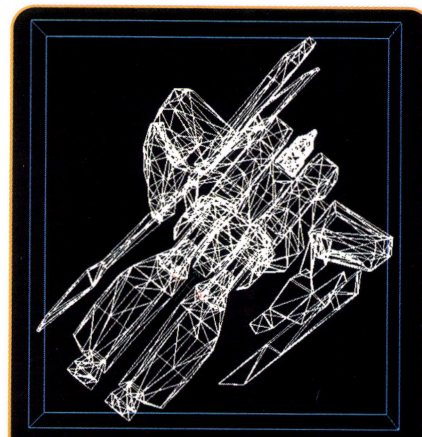


FIGURE 2. Axis-aligned bounding box on a rotated object.

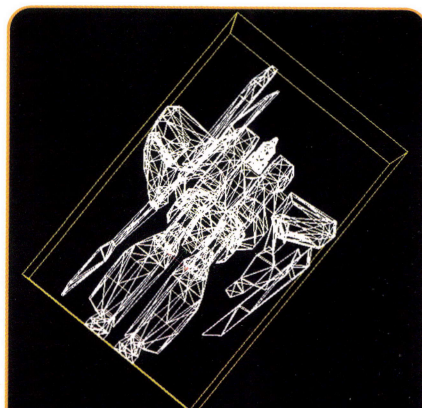


FIGURE 3. Oriented bounding box on a rotated object.

Jeff watches over a vast empire at Darwin 3D. He is also responsible for any collisions involving his e-mail box at jeffl@darwin3d.com. Test his collision response efficiency by sending him questions and comments.

the object into world space, given the current object orientation.

In the best case, this transformation can be reused when it comes to drawing the model. However, in the worst case, you'll be duplicating transformation work. In any case, the CPU is handling the matrix transformation for these bounding boxes. With the appearance of transformation hardware on graphics cards (such as the 3Dlabs Glint GMX and Diamond Fire GL 5000), this is a very costly operation. As this kind of hardware begins to dip into the consumer 3D hardware space, programmers need to be very careful to avoid using techniques that require vertex transformation by the CPU. Calculating a bounding box for a model with many vertices is a fairly expensive process. If your models are large, this can be a big frame-rate vampire that sucks the life right out of your game.

Getting More Bang For My Buck

You may wonder if there is any way to avoid having to transform every vertex into world space in order to find the bounding box. There is a way. However, like many things in computer game programming, there is a trade-off. Because I've already calculated the bounding box of the object in its rest position, it's possible to transform only those extreme points by the current orientation and get a new bounding box. In other words, I can take the vertices of the object's OBB and use the maximum and minimum of those positions to create a new AABB. I'm guaranteed that the new bounding box will completely contain the object because I've used the extreme extents of the initial position to create this new box. It may not be a tight fit, but it will fit. The good part is that this solution only takes eight transformations to calculate this new box — quite a bit of savings if your model contains many vertices. You can see the difference in Figure 4. The image on the right is the true AABB for the object. The image on the left is created using the yellow OBB as the source for the AABB. Listing 2 contains the code for calculating a bounding box this way.

Obviously, for some cases, this

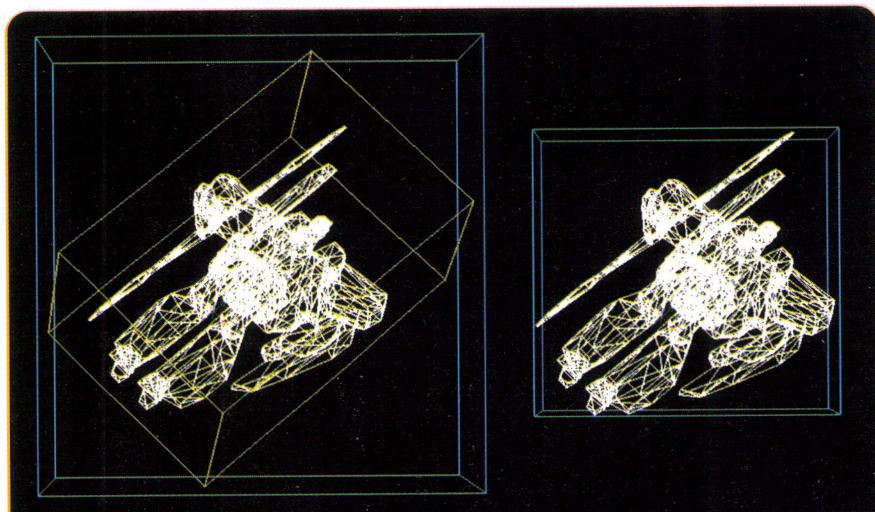


FIGURE 4. Fast and slow methods for calculating AABBs.

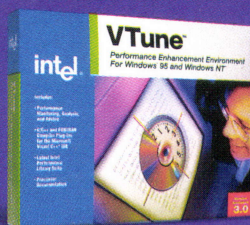
LISTING 1. Calculate an axis-aligned bounding box for an object.

```

////////////////////////////////////
// Procedure:      RecalcFullBBox
// Purpose:        Recalculates the BBox associated with a bone based on the
//                 new position for the vertices. Tighter fit in
//                 most cases. However, has to process all vertices
////////////////////////////////////
GLvoid COGLView::RecalcFullBBox(t_Bone *curBone, tVector *min,tVector *max)
{
    /// Local Variables //////////////////////////////////////
    tVector      *temp,tempRes;          // X,Y,Z VECTORS
    tNormalVertex *nvData;               // VERTEX WITH NX,NY,NZ,X,Y,Z
    t_Visual     *visual;
    //////////////////////////////////////
    visual = curBone->visuals;           // GET AT THE VISUAL ATTACHED TO A BONE
    nvData = (tNormalVertex *)visual->vertexData; // THE ACTUAL INTERLEAVED VERTEX DATA
    for (int loop = 0; loop < visual->faceCnt * visual->vPerFace; loop++)
    {
        temp = (tVector *)&nvData->x;    // POINTER TO THE VERTEX XYZ VALUES
        MultVectorByMatrix(&curBone->matrix, temp,&tempRes); // MULT BY THE BONE MATRIX
        // FIRST VERTEX, SET IT AS THE MAX AND MIN
        if (loop == 0)
        {
            memcpy(min,&tempRes,sizeof(tVector));
            memcpy(max,&tempRes,sizeof(tVector));
        }
        else
        {
            if (tempRes.x > max->x) max->x = tempRes.x;
            if (tempRes.y > max->y) max->y = tempRes.y;
            if (tempRes.z > max->z) max->z = tempRes.z;
            if (tempRes.x < min->x) min->x = tempRes.x;
            if (tempRes.y < min->y) min->y = tempRes.y;
            if (tempRes.z < min->z) min->z = tempRes.z;
        }
        nvData++;
    }
}

```


We know the territory. We understand the latest Intel® processors and know what each one **can** do. So for programs that fully utilize the Intel® Architecture, trust the people who **make** it in the first place. Intel's VTune™ Analyzer is critical in improving all of **your** hard work. The VTune Analyzer provides a system-wide view of all of your **code** and how it runs on Intel processors. Whether you're creating programs that **run** in C/C++, FORTRAN, or JAVA, they can execute a lot quicker. It's even **faster** than ever at pinpointing bottlenecks and eliminating any roadblocks, **buddy**.



*VTune™ Performance
Enhancement Environment*

Download a free 14-day evaluation at: <http://developer.intel.com/vtune>



© 1999 Intel Corporation. All rights reserved. Intel is a registered trademark and VTune is a trademark of Intel Corporation. All other brands and trade names are the property of their respective companies. Performance comparisons are between VTune Analyzer versions 3.0 and 2.5.

method doesn't result in nearly as snug a fit as our original AABB calculation. However, it's certainly faster, and on a system with hardware vertex transformation, you may prefer it. Which method is right for your game? How should I know? Try them both and see. If you're using bounding boxes as a quick early check and have more sophisticated methods in follow-up tests, this quicker method may be good enough.

When creating the initial bounding box for the faster AABB method, it's easiest to calculate the AABB of the object in its initial rest position. However, the initial bounding box doesn't have to be axis-aligned. You can achieve a better fit by defining an OBB in the modeling program when the object is built. The method will then use the OBB to calculate the current AABB. Depending on the model, an initial OBB may really help. Certainly, a box built on a diagonal is a perfect candidate for an initial OBB.

When Two Boxes Collide

My nice new bounding boxes overlap in two of my objects. Chances are, I have a collision occurring between them. But I have to make sure. Also, in order to do some interesting things, I need to determine exactly where they are touching. Take a look at Figure 5.

The bounding box of the two objects clearly are colliding. It's also just as clear that the objects themselves are not. This is why being a human is great and being a game programmer is difficult. A child can easily determine that the two objects in the picture are not actually hitting each other. If a game relies solely on bounding boxes for collision detection, players may feel cheated when two objects "hit" when it appears as if they didn't. So, how can the computer determine that this is really not a hit? Notice that both of these objects are convex. This distinction is very important. I'll talk about what to do if your object is concave later, but for now, let me only consider the case where the two objects being tested are convex.

Last month, I talked about the idea of separating lines. When dealing with 3D, if I can find a plane that separates

the two objects, then I know for certain that the objects are not colliding. I'll begin by considering each face of each object as the separating plane. It should be clear that the face highlighted in yellow in Figure 6 defines a plane that

completely separates the two objects.

In order to test this separation plane, I need to make sure that all the vertices of the right object are on the other side of this plane from the test object on the left. It's helpful to have a normal defined for each face in the model. If you don't have face normals defined in

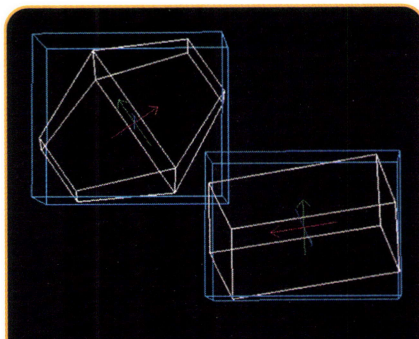


FIGURE 5. Two objects near a collision.

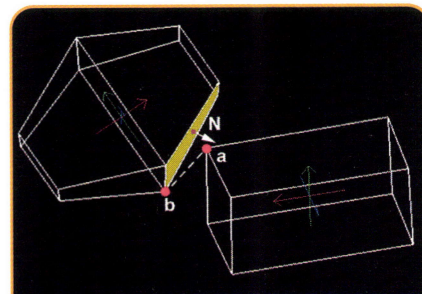


FIGURE 6. Finding a separating plane.

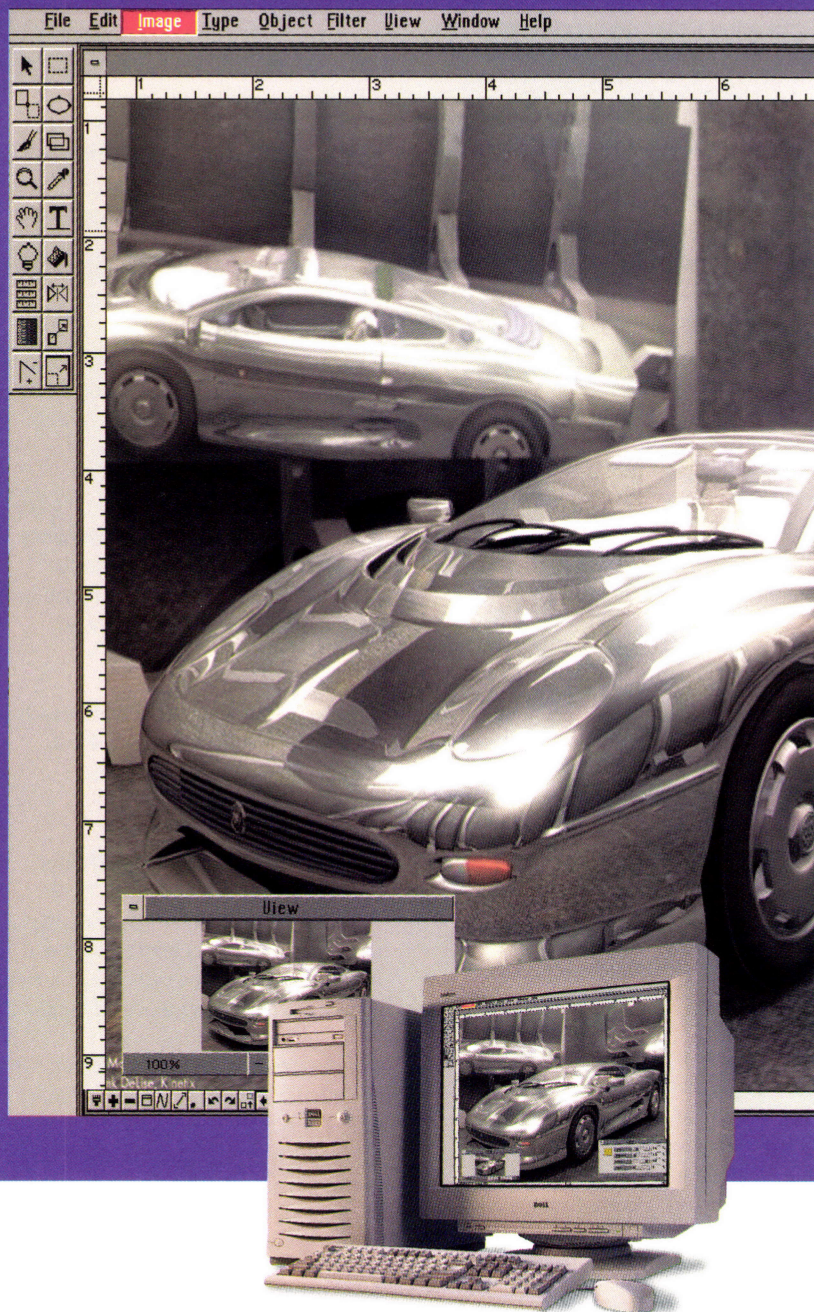
LISTING 2. Faster AABB calculation using starting OBB.

```

////////////////////////////////////
// Procedure:      RecalcBBox
// Purpose:        Recalculates the BBox associated with a bone based on the
//                  original bounding box. This is faster than the true BBox in
//                  most cases. However, this BBox is not as tight a fit.
////////////////////////////////////
GLvoid COGLView::RecalcBBox(t_Bone *curBone, tVector *min, tVector *max)
{
    /// Local Variables //////////////////////////////////////
    tVector    tempRes;
    int        loop;
    //////////////////////////////////////
    for (loop = 0; loop < 8; loop++)          // LOOP THROUGH ALL 8 BBOX COORDS
    {
        MultVectorByMatrix(&curBone->matrix, &curBone->visuals->bbox[loop], &tempRes);
        memcpy(&curBone->visuals->transBBox[loop], &tempRes, sizeof(tVector));
        if (loop == 0)
        {
            memcpy(min, &tempRes, sizeof(tVector));
            memcpy(max, &tempRes, sizeof(tVector));
        }
        else
        {
            {
                if (tempRes.x > max->x) max->x = tempRes.x;
                if (tempRes.y > max->y) max->y = tempRes.y;
                if (tempRes.z > max->z) max->z = tempRes.z;
                if (tempRes.x < min->x) min->x = tempRes.x;
                if (tempRes.y < min->y) min->y = tempRes.y;
                if (tempRes.z < min->z) min->z = tempRes.z;
            }
        }
    }
}
    
```


EVERY DAY OF YOUR LIFE YOU CREATE POWERFUL THINGS FOR OTHER PEOPLE.

(How about creating something powerful for yourself?)



Need a workstation with the power to match your creativity? Then you're ready for an Intel®-based Dell Precision™ WorkStation with Microsoft Windows NT 4.0, starting at \$2199. By building all systems to your exact specifications, we can give you some of the latest technology. Like single or dual Pentium® II Xeon™ processors. Intel's 100MHz front-side bus. Powerful OpenGL graphics. Expandable memory. And, of course, the great performance and reliability of Windows NT 4.0. And the Dell Precision WorkStation is certified to work with your specialized applications like 3D Studio MAX, Maya and Soft Image 3D. Not to mention a price that's bound to work with your budget. Call, or visit our website, to order.

DELL PRECISION™ WORKSTATION 610 NEW PENTIUM® II XEON™ PROCESSOR AT 450MHz (DUAL PROCESSOR/RAID CAPABLE)

- 128MB 100MHz ECC SDRAM
- 512KB Full-Speed Integrated L2 Cache
- 9GB Ultra-2/LVD SCSI Hard Drive
- 21" (19.8" viewable) Trinitron™ 1600HS Monitor
- 3Dlabs Oxygen GMX 2000 Graphics Card
- 40X Max™ Variable CD-ROM Drive
- Integrated Sound Blaster Pro Compatible Sound
- Integrated 3Com® Fast EtherLink™ XL 10/100 PCI TX Network Interface with Remote Wakeup
- Remote Client Manageability Support via DMI 2.0; Wakeup on LAN Capable
- **MS® Windows NT® Workstation 4.0 and 1-Year NT Telephone Support**
- 3-Year NBD On-site^A Service
- 7x24 Dedicated Workstation Hardware Telephone and Online Technical Support

\$5979

Business Lease*: \$220/Mo., 36 Mos.

 **E-VALUE CODE: 88652-490159**

Phone Hours: M-F 7a-9p • Sa 10a-6p • Su 12p-5p CT • In Canada,* call 800-839-0148
In Mexico,* call 001-800-210-7607 • GSA Contract #GS-35F-4076D

*Prices and specifications valid in the U.S. only and subject to change without notice. †For a complete copy of Guarantees or Limited Warranties, write Dell USA L.P., Attn: Warranties, One Dell Way, Round Rock, TX 78682. ^AOn-site service provided via service contract between customer and third-party provider, and is not available in certain remote areas. Technician dispatched if necessary pursuant to phone-based troubleshooting with technical support personnel. ^BBusiness leasing arranged by Dell Financial Services L.P., an independent entity, to qualified customers. Above lease payments based on 36-month lease, and do not include taxes, fees shipping charges; subject to credit approval and availability. Lease terms subject to change without notice. ^C40X Max/17X Min. Screen image courtesy of Dub Media, Rocket Science Games, Boffo Games. 3Com is a registered trademark and Fast EtherLink is a trademark of 3Com Corporation. Intel, the Intel Inside logo and Pentium are registered trademarks and Pentium II Xeon is a trademark of Intel Corporation. MS and Windows NT are registered trademarks of Microsoft Corp. Trinitron is a registered trademark of Sony Corporation. ©1999 Dell Computer Corp. All rights reserved.





**USE THE POWER OF
THE E-VALUE™ CODE.**

MATCH OUR LATEST TECHNOLOGY WITH OUR
LATEST PRICES. ENTER THE E-VALUE CODE
ONLINE AT WWW.DELL.COM/EVALUE, OR GIVE
IT TO YOUR SALES REP OVER THE PHONE.

TO ORDER TOLL-FREE

888-325-4272

BUILD YOUR OWN AND ORDER ONLINE

BE DIRECT™
DELL®
 www.dell.com

the model file, you can create them by averaging the vertex normals or by taking the cross product of two of the vectors that make up the face. I can begin once I have a face normal to test with — such as N in Figure 6. I create a vector between a vertex on the test face and each vertex on the colliding object. For example, I create a vector between points A and B in Figure 6 and call it Vector AB. Then I take the dot product of that vector and the face normal, $N \cdot \text{Vector AB}$. If this value is positive, vertex A is not colliding with the object. If all the vertices in the colliding object are on the far side of the separating plane, then I definitely don't have a collision, and I'm done.

What happens if I've gone through all of the faces and cannot find a separating plane? Do I definitely have a collision? Unfortunately, no. There may be a separating plane that's not a face on either object. An infinite number of planes exist, so how can I find one that separates the two objects?

Balancing on a Polygon Edge

Luckily for me, I don't have to try arbitrary planes to see if they separate the objects. It turns out that if the separating plane is made up of a face on the object, then it must contain an edge in the object. Figure 7 displays two objects that cannot be separated by any face in either object.

In this case, I create a plane composed of edge A and vertex B. If all of the vertices of Object 1, with the exception of those that make up edge A, are on one side of the plane, and all of the vertices of Object 2, with the exception of vertex B, are on the other side, then I may have found a separating plane. One last check has to be made in order to make sure that vertex B isn't actually on edge A. If they were collinear, that would obviously be a collision. Once that possibility is ruled out, I can mark this as the separating plane.

Once I've tried every edge/vertex pair as well as all the faces, then the objects must be colliding. However, once I've found this separating plane, I can be sure that the objects are not colliding and I can move on. After the separating plane is found, it should be stored. Going through all of the faces

and edges to find a plane is obviously pretty expensive to calculate. By saving the separating plane from the previous frame, I can take advantage of the fact that the separating plane tends to remain the same over several frames.

As I mentioned earlier, all of these tests only handle convex objects. It is much easier to determine collision on convex objects. If your objects are concave, you will need a method for creating a convex hull around the object. You can do this through a variety of methods. The concave object can be broken into several convex objects. There are also automatic methods of generating a convex hull around a concave object. One of the most popular methods is called QHull. You can find a link to it in the references. However, it may be much more efficient to have the artists create a convex collision object for every object in the game. That way, the artist can make the decisions about exactly what features are important to define as part of the collision boundary. This approach abides well by the game developer's philosophy: "Do as much work up front as possible, especially if it saves run time."

Boom, Boom, Out Go the Lights

That's all the time for this month. The sample application will allow you to load an object and play around with bounding boxes. When two objects have overlapping bounding boxes, a separating plane will be found, if possible. If not, a collision will be reported. Next month, I'll take up the issue of collision response so we can find out what to do once a collision has happened. ■

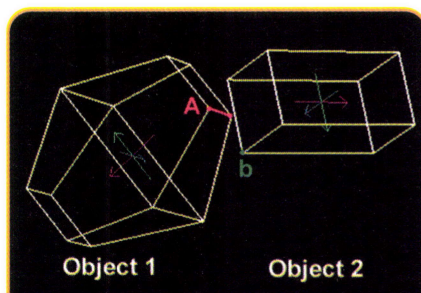


FIGURE 7. Separating two objects by an edge-plane.

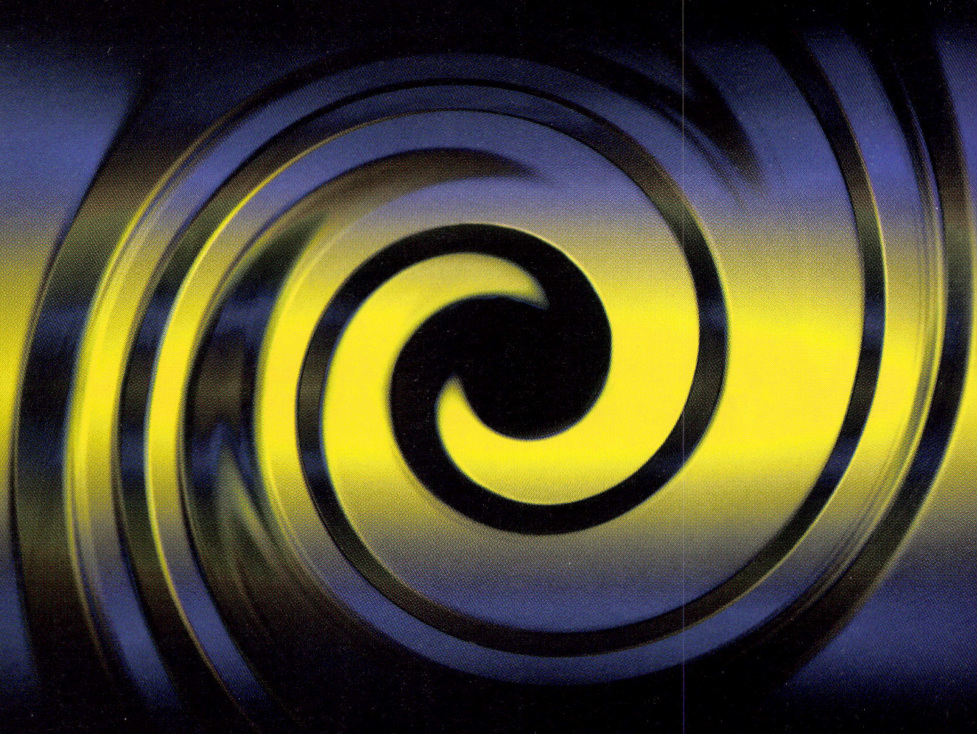
REFERENCES:

Collision detection is a very active area of research. This method represents only one.

- Gottschalk, Lin, and Manocha. "OBBTree: A Hierarchical Representation for Rapid Interference Detection." University of North Carolina, Chapel Hill. <http://www.cs.unc.edu/~geom/OBB/OBBT.html>
I mentioned the use of oriented bounding boxes. This paper extends this method to include a hierarchy of OBBs. Also, several sources discuss the use of tracking the closest features of two objects in order to determine when a collision occurs.
- Gilbert, Johnson, and Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space." *IEEE Transactions on Robotics and Automation*, April 1988, pp. 193-203.
- Lin, M. C. "Efficient Collision Detection for Animation and Robotics." Ph.D. Thesis, University of California, Berkeley, December 1993.
- Baraff, David, and Andrew Witkin, "Physically Based Modeling," *SIGGRAPH Course Notes*, July, 1998, pp. D32-D40. This paper also describes a method for using bounding box frame coherence to achieve more efficiency.
- Barber, Dobkin, and Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, Dec. 1996. This is the QHull library for calculating convex hulls. You can find more info at <http://www.geom.umn.edu/software/qhull/>
- There are also several libraries of collision routines available for use in non-commercial applications. Commercial applications may require a fee, so be sure to contact the source before using any library in a game application. These include I-Collide, V-Collide, Enhanced GJK, and others. A good link for all of these is at the University of North Carolina, Chapel Hill web site, <http://www.cs.unc.edu/~geom>.

IMMERSION '99

4th Annual 3Dfx Interactive Developer Conference March 15, 1999

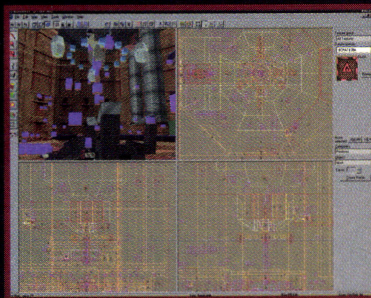
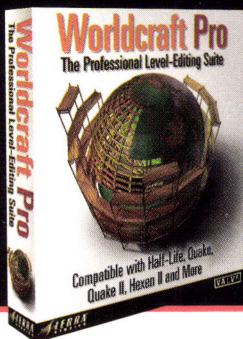


See www.3dfx.com for registration and conference details.

BEST HARDWARE. BEST SOFTWARE.

For Those Who Think Playing God Is A Good Thing.

Unleash your creative power with Worldcraft Pro, the professional level-editing suite for today's popular first-person shooters. As the only editor available for multiple titles—including Half-Life, Quake II, Hexen II, and other 3D shooters—Worldcraft Pro offers greater power and versatility than any other product in its class. And, while it is a professional tool—the very same used to create Half-Life—it also contains a user-friendly interface and tutorials that make it very accessible to the beginning designer.



This incredible level of Half-Life was built with Worldcraft Pro.



- ▶ The professional edition of the leading 3D level editor.
- ▶ No other level editor offers support for multiple titles.
- ▶ Worldcraft Pro's simple yet powerful interface makes it the ideal choice for beginning designers as well as professionals.
- ▶ Dedicated online community and level exchange center hosted by WON.

Worldcraft™ Pro

The Professional Level-Editing Suite

www.sierrastudios.com

© 1998 Sierra On-Line, Inc. All Rights Reserved. Sierra, Sierra Studios, Half-Life, Worldcraft, and WON are trademarks of Sierra On-Line, Inc. Valve and the Valve logo are a trademark of Valve LLC. Half-Life, Quake, Quake II, and Hexen II are trademarks or registered trademarks of their respective owners.

SIERRA
STUDIOS

VALVE

Playing God: Creating Convincing Environments in RT3D

In last-month's column, part one of the "Playing God" series, I outlined a process for streamlining the production of game-play-critical objects. This time around, we'll look at the problem on a larger scale. Instead of focusing on the creation of each individual model, we'll examine how to assemble the separate component

parts so that the result is a convincing, immersive world. We'll look at ways to populate the environment with vast detail (with minimal effort), as well as go over some general tips and tricks on creating the world itself.

The Art Bible

In January, we discussed the importance of the art bible as it pertained to game-play-critical objects. We saw how the art bible served as a guide to both designers and artists when trying to apply form to function and minimize the development

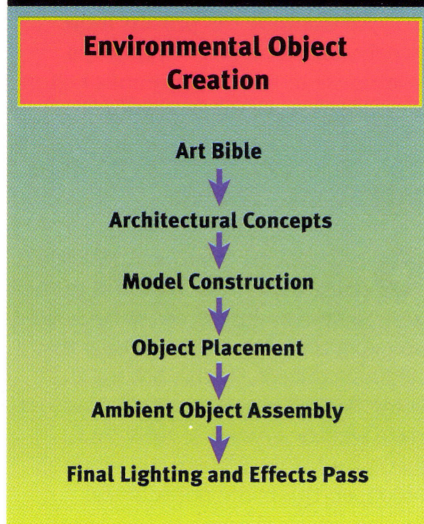
overhead while maintaining a coherent look-and-feel for game-play-critical objects. If the environmental art serves to define the aesthetic for the game, and the art bible outlines that aesthetic, it therefore follows that when it comes to creating convincing environments, the art bible remains the single-most critical document to which the artists will refer. Please see my column in last month's issue of *Game Developer* for more discussion about putting together a solid, evolving art bible. In short, a solid art bible should have enough information, organized intelligently, to completely familiarize new art team member with the game in only a few days.

floating islands suspended in the clouds. The important thing to remember is that you will be building your world literally from the ground up, so isolating a consistent technique early on is critical. Because the rest of your objects need to fit within this landscape, it is a good idea to spend extra time tweaking the style and composition, otherwise you may find yourself redesigning work that was based on a different aesthetic. The techniques for creating terrain are often specialized and unique to the rendering package being used, although with standardized hardware acceleration, the playing field is becoming increasingly level. This topic alone could fill an entire article, so we're going to assume uniform competency and move on to the next part of the process.

ARCHITECTURE. All artificial structures that game play takes place in or around constitute architecture. Usually, these spaces are separate from the world topography and can be developed concurrently. If the topography determines the overall atmosphere for the world, then the architecture determines its character. And because most game-play objectives depend on some sort of artificial structures to give them functionality, the architectural style for the world must be outlined before the game-play-critical items can be created.

AMBIENT OBJECTS. These are the little things that are used to fill out the environment. Though not critical to

FIGURE 1. Environmental object creation flowchart.



What Constitutes an Environment?

In the most general sense, environmental art includes all those resources necessary to create a graphically convincing world. It is critical to plan for and identify those items that are the minimum required for creating an immersive RT3D experience because time and personnel are both limited. These critical environmental items fall into three basic categories:

TOPOGRAPHY. Potentially the most time-intensive part of the project, this is the underlying structure within which all game play takes place. Depending on the genre of your game, the topography can be anything from a corridor-based underground maze to a string of

Mel has worked in the games industry for several years, with past experience at Eidos and Zombie. Currently, he is working as the art lead on DRAKAN (<http://www.surreal.com>). Mel can be reached via e-mail at mel@surreal.com.

game play, these items are essential in that they make the world believable. If the architecture objects in a level are analogous to a basic set of Legos, then the ambient objects in the level are analogous to the Lego expansion sets. Sure, you could do some pretty cool stuff with the basic set, but to get your

creations looking really cool and diverse, you need to bring out the expansion sets. Organizing the ambient objects in a way that is intuitive to use for artists and level designers can be a problem in itself, not to mention the mammoth task of creating all the objects themselves. To address this

problem we're going to be working with object databases.

Working with Object Databases

Creating a sufficient level of complexity in the environment is a task that can be tedious and time-consuming. One of the most efficient methods for creating detail in the environment is by means of object databases. An object database consists of a group of objects that can be used to create isolated pockets of detail, or to bring coherence to existing game-play areas. Let's say, for example, that you want to create several villages, each using the same style of architecture. The object database for the villages would contain three or four different buildings, plus an assortment of crates, barrels, wood-piles, curing sheds, outhouses, and so on. By using these objects in different combinations, a large degree of variation can be achieved with minimal overhead. The goal of the object database is to provide artists and level designers with a goody-bag full of generic building blocks with which to populate the world.

Figure 1 shows an example flowchart for the Environmental Creation Process. While these steps apply specifically to creating ambient game-play areas, the basic precepts can be applied to almost any part of the RT3D development process. The important thing is that you develop a process and stick to it. To examine the process more closely, let's look at an example, a farmhouse in the mountainous region of Surreal's world of *DRAKAN*.

Architectural Concepts

Once the basic architectural style is delineated in the art bible, each individual structure needs to be created on paper and then tested against the original concept. As the designs are being fleshed out, the art team can generate feedback for each individual piece. And since you are still working on paper, this is the point in the process where changes to the design can be made with rapid turnaround and little or no work loss. Figure 2 shows an example of a set of design sketches for human-based architecture.

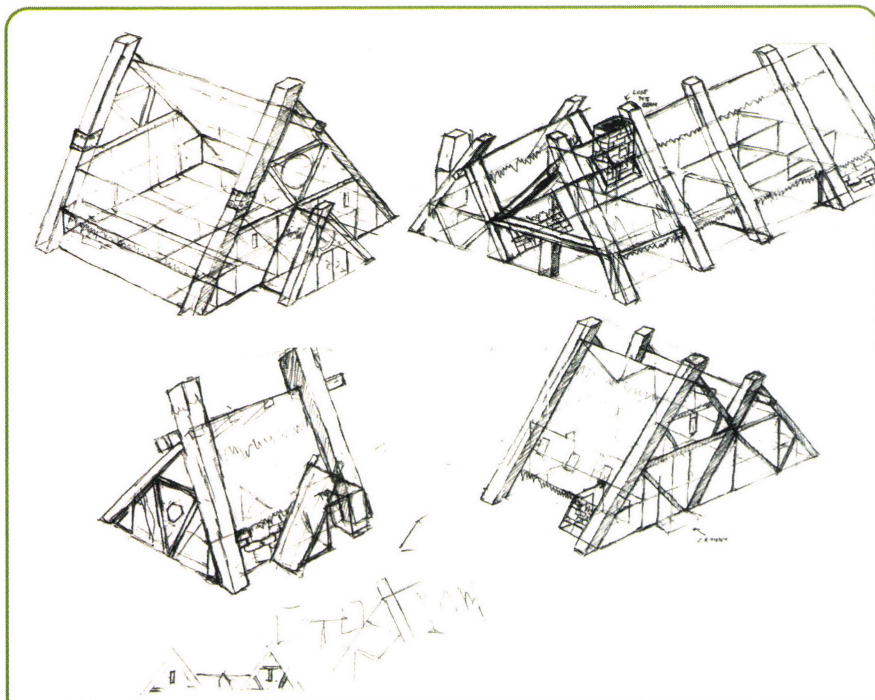


FIGURE 2. Architectural concept sketches from *DRAKAN*.

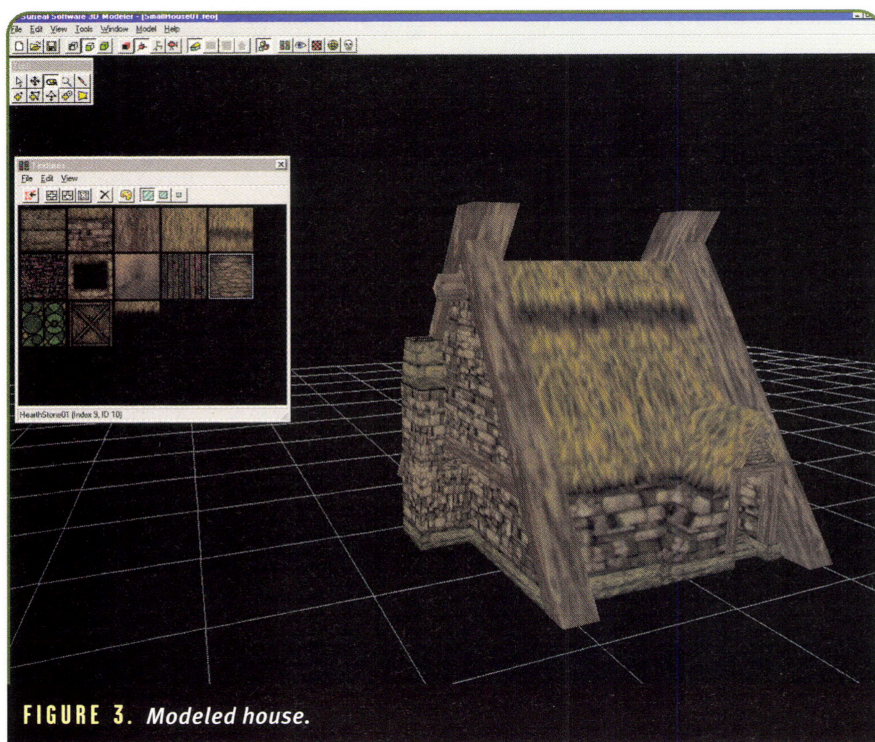


FIGURE 3. Modeled house.

Fun and games?

get serious

You've got a game in production and you're building the tools for the next one. **This is no time for playin' around.**

Get serious with industrial-strength tools from the Factory — CodeWarrior Professional and CodeWarrior for Playstation.

Here's why more than 100,000 programmers are dead serious about CodeWarrior:

- Cross targets: game consoles, PC, and Macintosh
- Wickedly fast build times
- Easy-to-use GUI interface
- State-of-the-art graphical debugger
- Support for AMD-3D™ technology

CodeWarrior®

www.metrowerks.com
1-800-377-5416
info@metrowerks.com



seriously



Game images courtesy of Ion Storm.

Model Construction

After the concept has been massaged and checked against the overall aesthetic, the structures can actually be built in 3D. Figure 3 shows the model of a farmer's house based on the previous architectural concept sketches. Note that there is little deviation from the concept sketch to the completed model. This is a good

benchmark against which to measure the effectiveness of your process. While each step brings an evolution of the aesthetic, if there are large style gaps between each step, you are probably not spending enough time in the early phases of the game. When a deadline looms, the processes tend to fall to the wayside due to time pressures. This can cause problems later on because as the aesthetic for each phase diverges from

the original concept, the results become less predictable, reducing your ability maintain a coherent artistic vision throughout the game.

Object Placement

In Figure 4, we can see the first pass at creating the farm. Note that the only things present at this point are the topography with its associated textures and the farmhouse. While the house is well designed and readily identifiable, the overall effect is not very convincing. The house lacks context. At this point it's time to go to the object database and begin assembling the individual pieces to add to the scene.

Ambient Object Assembly

In Figure 5, we can start to see the beginnings of a realistic outdoor scene. From our mountain world object database, some trees have been added to help achieve the effect of a mountainous forest. In this case, instancing and object redundancy have been exploited — there are, in reality, only two unique tree models in this scene, but they have been scaled and rotated to create variation and hide the similarities. Here again, we are getting the maximum amount of use out of our objects because we are using instancing. Note, also, that the farmhouse now has a better size reference, where before there was none. Still, the effect is not complete; the house appears in the scene as the only man-made object, and looks out of place.

Now it's time to dip into the farmhouse object database goody-bag and customize our farm. To do so, we will add four objects: a barrel, a crate, a cart, and a section of fence. With the addition of these ambient objects, the farmhouse is unmistakable (Figure 6). Not only do the ambient objects serve to give variation and diversity to each individual structure, the additional objects serve to cement firmly the structure's place in the landscape, while filling out the scene with interesting eye candy. This aids in game play as well because now that there is a fence, the player is funneled directly



FIGURE 4. Structure placed in landscape.



FIGURE 5. Farmhouse with trees.

MultiGen Creator

The Complete Realtime 3D Toolset

Join The Hitmakers

A partial list of hit titles built using MultiGen-Paradigm tools:

Atari Games/Midway

San Francisco RUSH™ (N64, Arcade, PC, PCX)

Rare Ltd.

GoldenEye 007™ (N64)

Banjo Kazooie™ (N64)

Twelve Tales: Conker 64™ (N64)

Zipper/EA/Hasbro

MechWarrior III™ (PC)

Top Gun: Hornet's Nest™ (PC)

Recoil™ (PC)

BlueShift, Inc.

Vapor TRX™ (Arcade) Atari Games

Nintendo

Super Mario™ (N64)

Zelda™ (N64)

Origin/EA

Longbow II™ (PC)

Jane's F-15™ (PC)

Sierra On-Line, Inc.

Viper Racing™ (PC)

SingleTrac

Twisted Metal™ I, II & III (PSX & PC)

Jet Moto™ I & II (N64)

Warhawk™ (PSX)

Check Out MultiGen Creator

Contact us to request a free evaluation of MultiGen Creator. Visit the MultiGen-Paradigm web site to download the OpenFlight file format and get lots of useful realtime information

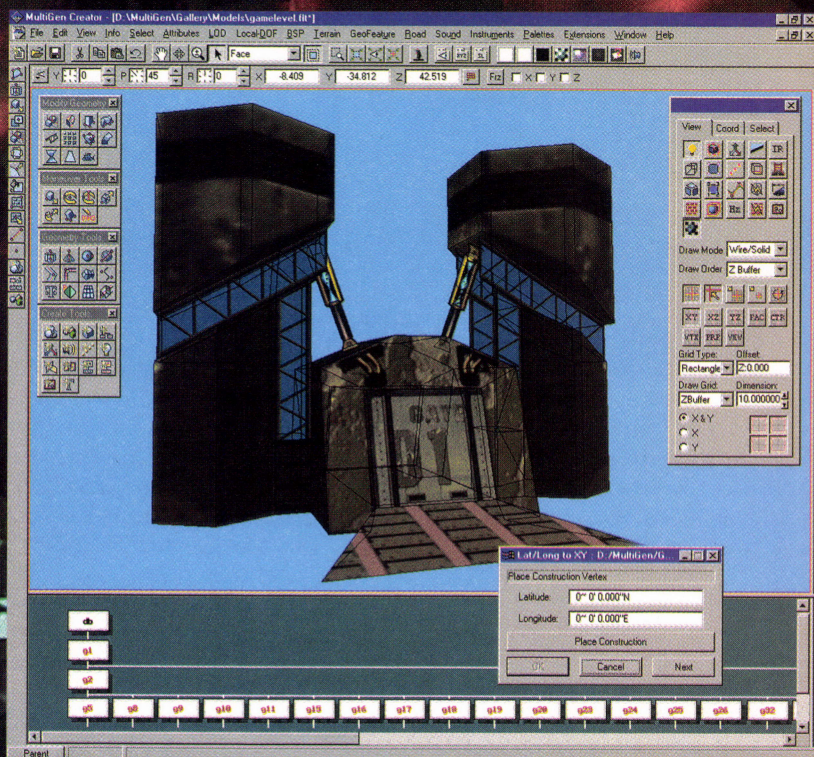
Contact us for more details

Phone: 408 261 4100

Fax: 408 261 4103

email: sales@multigen.com

http://www.multigen-paradigm.com



HIGHLIGHTS OF MULTIGEN POWERTOOLS

- Import Art from 3D animation tools
 - 3D Studio MAX
 - SoftImage
 - Alias
 - Nichimen . . . Many More
- Game Data Tagging and Editing
- Automated Level of Detail (LOD)
- Hierarchy Structure Editing
- Binary Separating Planes (BSP)
- Precise UV Texture Mapping/Editing
- Texture Rendering Control
- Damage and Switch States
- Degrees of Freedom (DOF)
- Collision and Bounding Volumes
- Instancing and External Referencing
- Automated Terrain
- Automated Roads and Pathways
- Scene Optimization
- World and Level Assembly
- Compatible with Multiple Realtime Formats
 - Openflight
 - VRML
 - NIFF
 - HMD
- Extensibility & Customization
 - Data Read/Write API for converters and loaders
 - Data Extensions API for custom game data
 - Plug-in API for custom tools
- Multimedia Asset Management

Images from Recoil™ courtesy
Electronic Arts & Zipper Interactive, Inc



MultiGen-Paradigm
THE REALTIME 3D COMPANY

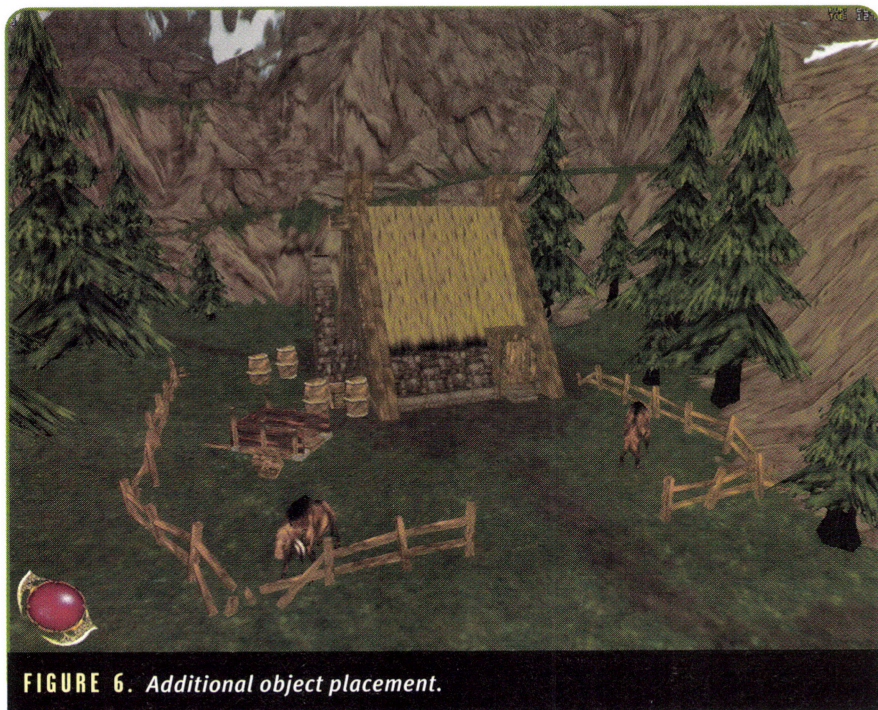


FIGURE 6. Additional object placement.

towards front side of the house. Finally, with the addition of a cow or two, the scale of the building becomes even more recognizable, so that the overall effect is very convincing.

Final Lighting and Effects Pass

Once you've got all the models assembled in your world, it's time to add the finishing touches to the area to give it some atmosphere (Figure 7). We've adjusted the lighting in the world to mimic the ruddy glow of a setting sun, with a small amount of blue in the shadows. We've also added a small campfire in front of the house to give it some additional character. Most engines that support hardware accelerators also support colored lighting, so it's a good bet you'll be able to get your scene looking something like this. Now, go back and compare this result with the image in Figure 4, and you can see that with just a little preparation and attention to detail, a few objects can make a huge difference.

Tips and Tricks

TEXTURES AND LIGHTING. When creating textures that are going to be lit with in-game lighting, you'll need to be careful to avoid textures that are too dark.

Most lighting routines slightly modify the existing RGB values of the texture, which boosts the already existing hues. This means that dark textures tend to stay dark, which basically defeats the lighting process.

LIGHTING. As we demonstrated in the above examples, taking the time to light your levels with dynamic color combinations can yield a tremendous return on the time invested. However,

this does take time to get right, so do some experimentation and schedule yourself enough time to tweak it.

OBJECT DATABASE USAGE. One of the best ways to take advantage of the object database is to have a junior level designer or artist simply go over the levels and drop in objects where they see fit. This is a good experience for the artist or designer and it doesn't tie up your key people on a job that is somewhat tedious and mundane.

USE MOCKUPS WHERE POSSIBLE. If your team is coding their own engine, there will probably be some dead time until the art path is solidified and you can actually get your resources into the game. In the mean time, try setting up your environments in whatever modeling package you're using in-house (Softimage, 3D Studio Max, and so on). This works especially well when trying to come up with special effects routines in whatever engine you're using. By sitting down with an engineer and playing around with the particle system in your modeling package, for example, you can work together to visualize the effect you're trying to achieve before any time is wasted moving in the wrong direction. ■

SPECIAL THANKS:

Alan Patmore, Hans Piwenzky, and Louise Smith.



FIGURE 7. Final lighting and polish.

San Jose Convention Center | Expo: March 16-18 | Conference: March 15-19

Nihilistic is bringing the entire team to the GDC '99.

(They may have to take three cars.)



nihilistic software

L to R Back Row:

Yujin Kiem | Artist
Maarten Kraaijvanger | Lead Artist
Rick Sorvande | Artist
Steve Tietze | Director of
World Design | Lead Level Designer
Rob Huebner VP | Director of
Technology | Lead Programmer

L to R Front Row:

Brent McLeod | Level Designer
Steve Thoms | Level Designer
Ingar Shu | Sound Designer |
Programmer
Ray Gresko | President | Producer |
Designer | Programmer |
Business Manager
Yves Borckmans | Programmer
Anthony Chiang | Technical Artist

Even seasoned, professional game developers with successful titles under their belts come back to the Game Developers Conference year after year. They know that in a fast-moving, competitive industry like ours they must stay on top of new technologies and techniques. At the GDC they learn from over 250 industry experts and thousands of their peers. Whether your team is developing for the PC, console, arcade or location-based entertainment, a successful production cycle starts with a week at the GDC. That's where you'll find serious sessions, tutorials, and networking opportunities for every team member. And an exhibit floor filled with the latest tools and services. See you in San Jose!

The Game
Development Platform
for Real Life

GAMEDevelopers
C O N F E R E N C E 1 9 9 9

San Jose Convention Center | Expo: March 16-18 | Conference: March 15-19

(To Register)

w: www.gdconf.com/mgdm | t: 888.234.9476

f: 415.905.2222 | e: register@mfgame.com

Miller Freeman Game Group:

Game
DEVELOPER

Gamasutra.com

GAMEDevelopers
C O N F E R E N C E

3Dfx Against the Rest

No graphics chip or board company has reigned supreme over its competitors for very long. Usually, once a company achieves the exalted position of Number One, it can look forward to a short stay at the top. And no graphics chip or board company has built as strong a consumer brand,

in as short a period of time, as 3Dfx. In 1999, 3Dfx will have to transition from a 3D-only game players' brand to a fully fledged graphics company that competes in both OEM and retail markets. How 3Dfx competes, and how the competition behaves, will define the landscape of 3D graphics for PC game developers.

Voodoo3: A Miss or a Direct Hit?

At Fall Comdex 1998, 3Dfx announced Voodoo3. In many ways, Voodoo3 is what Banshee could have been, and in some ways it's a descendant of Voodoo. However, Voodoo3 certainly isn't a clear-cut winner for the company because it puts 3Dfx in the same 2D/3D graphics chip category as everyone else in the business.

First, Voodoo3 is not one chip or chipset. It comes in two packages: the 2000 and 3000 versions. The Voodoo3 2000 is targeted at the OEM market, while the Voodoo3 3000 is a more high-

end version targeted at the retail market where Voodoo and Voodoo2 have done so well. The differences between the two Voodoo3 chips and Banshee are quite straightforward. The Voodoo3 has the dual texture engines of a Voodoo2; it supports AGP 2X, with planned support for AGP 4X in the works. 3Dfx claims Voodoo3 delivers performance in excess of Voodoo2 SLI configurations, but in a 2D/3D integrated package. The Voodoo3 3000 is the much more interesting part of the two; it's capable of supporting resolutions of up to 2048x1536 at 75Hz refresh rates. However, Voodoo3 is an 8.2 million transistor titan of a chip. That makes it as complex as any high-end CPU on the market today, and 3Dfx will have its work cut out to deliver the highest performance versions of the chips by its mid-1999 deadlines. The most likely scenario is that the Voodoo3 2000 will be the first chipset to ship in full production, followed by the Voodoo3 3000. It's also likely that in the retail

channels, you will find both versions of the chipset, although PC OEMs probably won't want to integrate the Voodoo3 3000 because of its higher cost.

While Voodoo and Voodoo2 gave 3Dfx a unique position in the retail market, the emergence of Voodoo3 puts them squarely in the same market as any number of other chip vendors. At one point, they could have remained aloof from the fray, but now 3Dfx must duke it out in the same mud pit with everyone else. It's a no-holds-barred contest that 3Dfx may not be equipped to win, even with the best technology.

The Competition

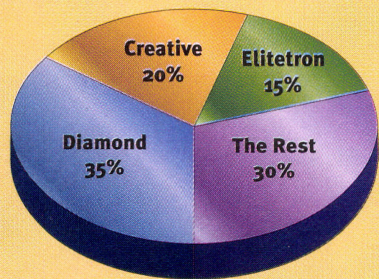
In direct competition with 3Dfx are Nvidia and 3Dlabs. These two companies compete via technologies that are supposed to drive their products into the mainstream from the upper end of the multimedia PC spectrum. What these companies covet most is the

*3Dfx's quarter by quarter growth. In 1996 the company made in excess of \$6 million. *The company noted \$13 million as income, primarily due to its successful court case against Sega and NEC/Videologic. (Source: company financial statements; figures are in thousands).*

	Q4 1996	Q1 1997	Q2 1997	Q3 1997	Q4 1997	Q1 1998	Q2 1998	Q3 1998
Revenue	\$4,503	\$5,247	\$6,507	\$10,018	\$22,296	\$50,008	\$58,643	\$33,206
Cost of product	3,404	2,582	3,278	5,352	11,399	25,730	30,443	24,971
Gross profit	1,099	2,665	3,229	4,666	10,897	24,278	28,200	8,235
Research and development	2,288	1,953	2,397	3,201	4,860	5,826	8,308	10,038
Selling, general and administrative	2,422	1,846	2,521	2,684	4,338	9,638	8,041	6,971
Net income (loss)	(\$3,598)	(\$1,161)	(\$1,753)	(\$872)	\$2,704	\$7,462	\$9,032	\$3,118*

Omid Rahmat works for Doodah Marketing as a copywriter, consultant, tea boy, and sole employee. He also writes regularly on the computer graphics and entertainment markets for online and print publications. Contact him at omid@compuserve.com.

Board makers' shares of 3Dfx revenues for the first three quarters of 1998.



slot occupied by companies such as ATI and Matrox, but the technological area that they have gone after is the 3D performance of Voodoo. By contrast, Matrox, despite some poor showings in the 3D performance stakes, continues to dominate PC OEMs product segments where 2D performance is still the primary driving force. As a result, Matrox has a strong position in the entry-level Windows NT workstation lines of Compaq and Hewlett-Packard, product lines that are squarely aimed at financial markets and standard corporate users. Both ATI and Matrox, however, have enough branding to make it onto most of the major retail shelves and, as a result, they have a strong consumer following. S3, Trident, and even Intel (with its agenda to sew up the low-cost PC market) don't possess the same performance advantages of 3Dfx or Nvidia. These companies all need the volumes that PC OEMs bring in order to fuel future developments — and their ambitions. In the midst of all this, PC OEMs just want to make sure that sockets are filled in the confusing array of product segments and lines they produce.

3Dfx and Nvidia found some solace in the fourth quarter of 1998 when Matrox, ATI, S3, and 3Dlabs failed to deliver products competing products. By the time Voodoo3 is supposed to come out, the competition may be stronger. S3 has plans for a follow up to Savage3D, ATI is expected to be in full swing with the Rage 128, and 3Dlabs could still make it with Permedia 3. On paper, all these chipsets have an equal chance of succeeding against the others, and in truth, if they all work as they should, they will all find a home somewhere. No PC maker wants to sell the same components as a competitor, so there's plenty of business to go around.

3Dfx stands to gain the high ground if — much as Nvidia has to date — it can get its foot into both corporate and consumer multimedia systems on the basis of performance, reliability, and support.

Creative and Diamond at the Door

Another threat looms for 3Dfx. An outside chance exists that 3Dfx will merge or be acquired by a third party. So goes the general thinking in the industry, at least. The reason for this speculation can be traced to attempts by Diamond and 3Dfx to form a joint venture in late 1998. The story goes something like this: Diamond and 3Dfx started discussing ways of binding themselves to each other. The rumor mill spoke of mergers and joint ventures, but we may never know for sure. What is known is that Diamond initially benefited significantly from its relationship with 3Dfx, but then saw its profits from sales of Voodoo and Voodoo2 get eaten up by the force of numbers of other 3Dfx add-in board customers, particularly Creative Labs. Creative, in turn, got wind of Diamond and 3Dfx's machinations and did what any company with deep pockets would do: it voraciously consumed 3Dfx shares, making sure that any chance of a Diamond and 3Dfx partnership would raise the specter of its own bid for the company. That's the story, but it doesn't end there. Diamond knows that it needs technology and intellectual property to succeed. Creative Labs doesn't want Diamond to get its way. As a result, all chip companies are up for grabs at the right price, and consolidation in the 3D consumer graphics arena is only a matter of time. 3Dfx just happens to be the juiciest target.

Entertainment: Make it Less Fun?

3Dfx has transcended the image of other graphics chips, and firmly embedded itself in the minds of consumers. With so much brand equity built among end users, 3Dfx could quite easily do away with the middleman and sell boards into the market directly. However, there are some issues that, from the company's perspective, diminish the possibility of such an event.

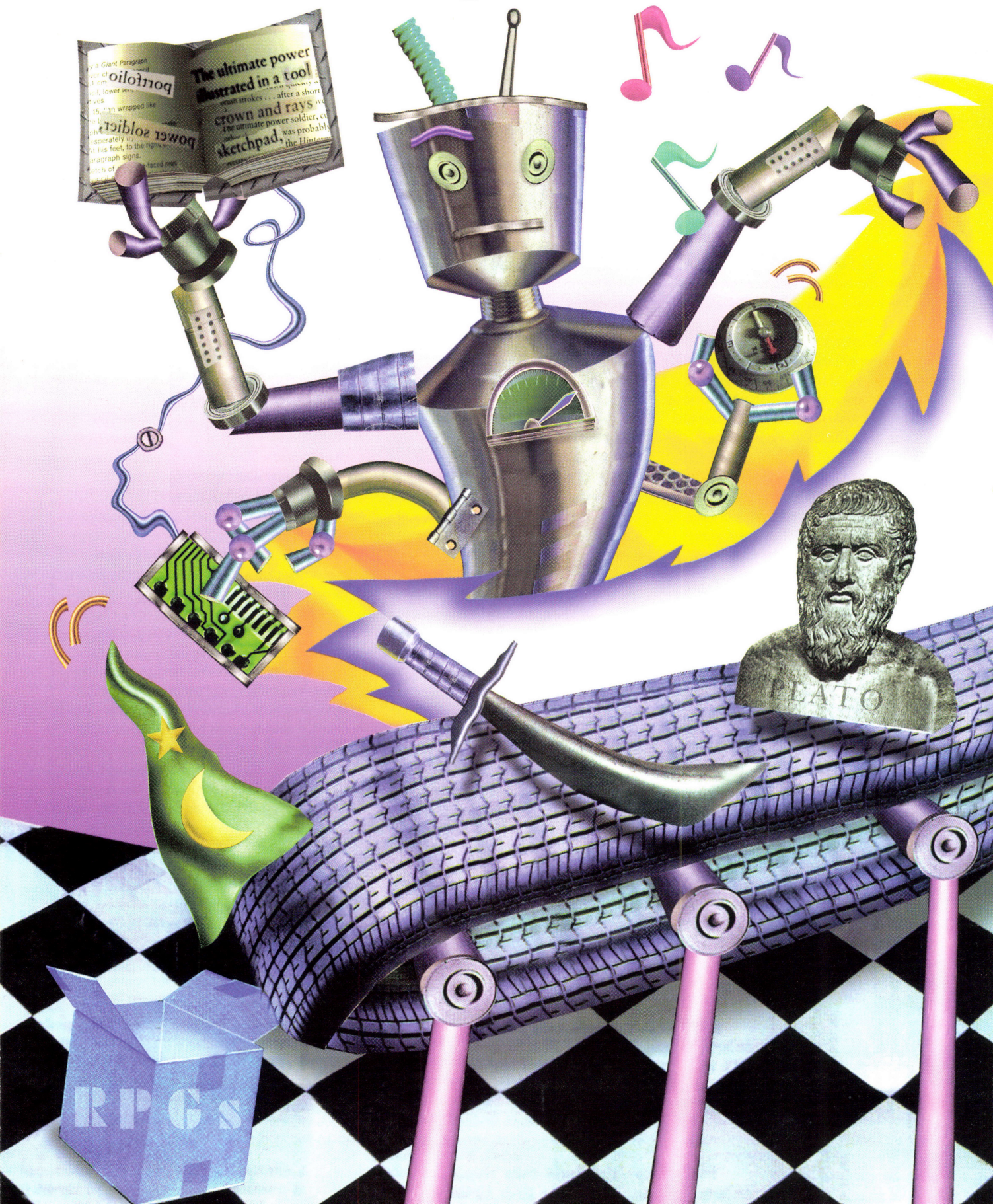
First, graphics chips still deliver better

profit margins than boards. True, you can make more money with boards, but there is also the question of support, and sales and marketing through indirect channels (retailers and distributors). Presently, 3Dfx is focused on its technology and is leveraging its brand among game enthusiasts to drive its retail presence and with strong brand recognition. PC OEMs continue as customers. In the final part of 1998, only Nvidia and 3Dfx were setting the world on fire with new graphics chips. It's a sellers' market for now, and game players' appetites for 3Dfx products — diminished though they may be — are still sufficient to keep the momentum going.

News Flash

As this article was going to press, 3Dfx announced its intention to acquire STB Systems, and to become a fully-fledged provider of graphics chips and boards. This deal is expected to go through by March 1999. Among the benefits of this deal (as told by 3Dfx) are providing PC OEM and retail customers a single source for 3Dfx branded add-in boards for greater price stability and smoother product transitions; the provision of reliable manufacturing capabilities as prequalified supplier to the top ten PC-OEM manufacturers; a more tightly integrated chip, software, and board-level layout and design for faster time-to-market and the most cost-effective graphics solutions; and a more controlled and focused brand investment for an even stronger retail presence. This stirs things up big time in the graphics industry, and confirms what the industry believed would happen to 3Dfx. The outlook for Creative and Diamond is not clear, and probably by the time you read this Nvidia and other chip vendors will have made their own strategic moves as a counterpoint. So, now we have ATI, Matrox, 3Dfx/STB, 3Dlabs/Dynamic Pictures, and Evans and Sutherland/AccelGraphics as chip and board vendors against Creative and Diamond as brand name board makers with chip agnosticism. Not too long ago, there was really only ATI and Matrox. So now, as far as game developers are concerned, the board is the platform, and whoever has the market clout reaps the accelerated titles. Should be fun to watch. ■

REMODELING



RPGs FOR THE NEW MILLENNIUM

BY WARREN SPECTOR

Despite working in a truly remarkable medium, one that provides powerful tools for the simulation of fantastic worlds and myriad ways to immerse players in them, many RPG designers these days seem content to recreate the glories of earlier computer games. Worse, many designers seem content to recreate experiences that they (and we, as players) first enjoyed in other media.

As I stated in my recent Soapbox column ("It's ROLE-playing, Stupid!", September 1998), we RPG designers are setting our sights too low. Look at the best RPGs of the last several years. As great as *DIABLO*, *FALLOUT*, *DAGGERFALL*, and *MIGHT & MAGIC VI* are, they really aren't anything that we couldn't have designed ten years ago. Do they represent significant advances over *WASTELAND* or *ULTIMA IV* or the *UNDERWORLD* games? And were these older games striving for much more than a recreation of the tabletop role-playing experiences of their creators? It's as if we can't see beyond our early *Dungeons & Dragons* game experiences. It's time to move beyond simply borrowing game concepts and establish computer RPGs as an independent medium.

Warren Spector runs Ion Storm's Austin, Texas, office. He is currently working on a new role-playing game, *DEUS EX*. In the past, he has produced such RPGs as *ULTIMA VI*, *ULTIMA VII*, *PART 2: SERPENT ISLE*, *UNDERWORLD 1*, *UNDERWORLD 2*, and *SYSTEM SHOCK* for Origin and Looking Glass Technologies. You can reach him at wspector@ionstorm.com

Illustration by Robert Zammatelli

My intent with this article is to lay out the abundant variety of choices available to would-be and practicing RPG designers. Only by analyzing the tools that we all use in the creation of our games, discussing the ways in which these tools have been and can be used, and identifying the ramifications of those uses, can we take this genre forward.

It's probably not an overstatement to say that every new medium begins, creatively and aesthetically, by defining

itself in terms of previous media. So it's not surprising that we RPG developers borrow from our forebears. In fact, every RPG developer today really owes his or her job to Dave Arneson and Gary Gygax, the creators of *Dungeons & Dragons*. Our debt to these two men is worth noting here if only to force a recognition of how little we've moved beyond the realm of 20-sided dice, the concept of character class, and those familiar core attributes of strength, wisdom, dexterity, and so on.

How do we identify a computer RPG? For the purposes of this article, a computer RPG is a game in which character development and character interaction take precedence over other factors and where each player's experience of the story is determined by individual choice rather than designer fiat. Though broad, this definition clearly eliminates real-time and turn-based puzzle and strategy games (lack of character development and interaction), as well as shooters and platform action games (lack of individual choice). Of greatest importance, this definition eliminates adventure games, which share with the RPG an emphasis on story and character. What adventure games lack — and this is a critical point — is the capability for players to grow and develop their characters, and to affect, if not the outcome of the story, than the way in which the story unfolds. Without both character development and genuine choice placed within a player's control, a game cannot be called a role-playing game, as I choose to define the genre.

Someday, we will concentrate on those aspects of computer RPGs that set them apart and we *will* leave our paper gaming roots behind. To do so, we have to be more daring in our designs — or, in the terminology of this article, in our selection of design tools. And to select the right tools, we need a better understanding of these tools and how they define the genre.

The term "tool" seems, at first, odd to use in the context of game design. When I use this term, I'm talking about the conventional *elements* that are sure to appear in any work that defines itself, or is defined, as a computer RPG. Any designer contemplating an RPG must take a stance with regard to all of these tools, even if that stance is to deemphasize one or more of them. What follows are the defining characteristics that must be present in any RPG.

Story

RPGs are story driven. There's a reason for talking to or killing people and monsters, a reason to build or destroy things. Unfortunately, though it goes without saying that

RPG Commandments

1. Each player's path through the story must be unique. This doesn't mean a branching-tree structure with winning and losing paths but, rather, that players will have the freedom to decide how they'll overcome game obstacles. A world simulation must be deep enough so that each game problem is open to a variety of solution strategies, from the most thoughtful and low-key to the most obvious and violent. And the solution you choose to any given problem must have clear consequences, both immediate (killing a guard sets off an alarm, attracting more guards) and long-term (killing a guard may result in "wanted" posters being posted, causing civilians to fear you and be less cooperative).

2. Players must always have clear goals. Though free to stray from the storyline at will, players must know what they're supposed to be doing, minute to minute and, if appropriate, mission to mission. The fun of the game is in overcoming obstacles and solving problems; the fun is in *how* you solve a problem, not in guessing *what* problem you're supposed to solve.

3. The level of interactivity must be high, with NPCs about whom you really care and with a densely populated, object-rich world that looks and behaves like the real world (or, at least, a believable, internally consistent world of your own creation). A big, empty world is boring. Players must be free to explore a cool and instantly understandable world.

4. The central character must grow and change in ways that matter to players in an obvious and personal way. During the course of play, you'll become more powerful, acquire more items, and develop new skills, of course. However, you'll also make unique friends and enemies, accomplish tasks and missions differently, overhear different conversations, and see different events unfold. By game's end, each player must control an alter ego that is distinct from that of all other players.

5. The game must be about something more than killing things, solving puzzles, and maxing out a character's statistics. Remember all those hours you spent in school analyzing the underlying meaning of novels, poems, and movies? Guess what: RPGs lend themselves to the same kind of analysis. Games can and must have an impact on players. That impact may be the simple adrenaline rush of *DIABLO*, fleeting and soon forgotten (nothing wrong with that), or it may be the never-to-be-forgotten (and, in some cases, life-changing) experience of becoming the Avatar in *ULTIMA IV*. If all you're doing is throwing wave after wave of monsters at players so that they can kill lots of stuff so that they can increase some arbitrary statistics so that they can feel powerful, you're doing yourself, your players and your medium a disservice.

RPGs must tell a tale, it's unclear whether the quality of that tale has much, if anything, to do with a game's success. One would be hard-pressed to describe the Avatar vs. Guardian (a.k.a. "kill the evil foozle") stories of recent ULTIMA games as on a par with what we demand from books and movies. DIABLO's plot hardly qualifies as compelling. UNDERWORLD's story of a hero locked in a dungeon until he can rescue a kidnapped princess hardly qualifies as narrative genius.

Currently, the kinds of stories we can tell seem to be limited by the expressive capabilities of our medium — it's tough to tell a great story when you can't recreate a young lover's shy smile or allow players to tell a joke rather than bludgeon somebody. Right now, what we do most easily and best is direct, one-on-one conflict (typically combat interactions), a fact that limits our narrative range just a tad. This is not to say we shouldn't strive for greatness in our stories, but we must find greatness in the strength of universal themes and in the ways in which we tell our necessarily simple stories.

Players of RPGs must have some degree of freedom in how they follow the threads of the plot and, in some cases, how the plot resolves itself. They can often pick the order in which they accept quests or even which quests they take and which ones they ignore. Further, how they conduct themselves during a quest, and how they interact with other characters, can alter the course of the story and its outcome.

The first and, arguably, most critical decision RPG developers must make with regard to story is whether to use a branching structure or to tell a story in a more conventional, linear fashion. The temptation is strong simply to say, "We're making a computer game. Computers allow branching in a way and at a level no other medium allows. Of course, we should use a branching structure." This argument, one I've made myself, goes back to the moral imperative to maximize the unique capabilities of the medium and to turn away from the techniques more appropriate to other media. It's perfectly understandable that computer RPG developers would want a branching structure if for no other reason than to differentiate games from books and movies. But let's

think through the implications of that decision.

Often, making one choice — picking one branch over another — means that a player can't go back to the branch not chosen. If I may be prescriptive for a moment, if picking a branch does *not* limit players' later options in some way, the branch is unnecessary and a waste of valuable development time. The illusion of player freedom isn't worth the development price.

However, assuming branching offers real choices (meaning, choices that limit player options even as the player moves forward through the plot), the approach can be worth the cost. Done well, branching can provide a powerful illusion of freedom for players. But, that's all it can provide — an illusion. The reality is that, if we don't put something in the game, on the screen, in the mouths of nonplayer characters (NPCs), it doesn't happen — and no amount of branching can allow players to do things we don't allow them to do. What this means is that the choices available to players solely as a result of branching are false, because eventually players are forced back onto one of the paths that we've created for them.

The first factor to consider when assessing whether branching is appropriate and/or necessary for your project is whether it's worth sinking valuable development resources into the creation of content that many, if not most, players will never see. And bear in mind that you're going to be spending time and money to ensure that the game makes sense regardless of the order in which each player sees each portion of the story. That's a lot of extra flags to set and check and a lot of extra art to create on the off-chance that players will stray from the logical path.

But what about replayability? Doesn't branching encourage players to keep playing a game? My first response would be, "Nah. By the time they finish your 100-hour epic, they're probably looking for The Next New Thing." Only the most zealous players replay games at all, and they're sure to see that a big percentage of their adventure differs not at all on subsequent playthroughs no matter which plot branches they follow.



None of this is to say that branching isn't worth all that

extra effort. Though not vital to success (aesthetic or commercial) it's important that players talk about their experiences playing your RPG and, when they do, it's powerful when their descriptions differ, seemingly based on individual choices. As in all development-oriented decisions, it's important to weigh that power against the cost of achieving it.

It's also important to realize that once you do spend your development dollars on giving the player power over the way in which your story unfolds, that should become the emphasis of your game. You should try to give your players a big, contiguous world to explore and you should let them explore it freely and in any way they want — even at the expense of character development.

The alternative to branching is to tell a more traditional linear story. But telling a story in the way that stories have always been told isn't the answer. So what are the advantages of telling a linear story and how is this best achieved? Let's start with the biggest and most obvious advantage of the linear narrative, the story itself.

Clearly, you can tell a better story if you don't have to worry about and/or deal with all the ways in which players can screw up your carefully crafted epic narrative. It's generally accepted that a linear story in a game almost inevitably means a more powerful story. Given the cost of achieving the illusory freedom offered by branching storylines, the linear story seems to be a pretty good deal. In addition, depending on how you implement your linear story, you may find it possible to give players some genuine freedom to personalize their experience



FIGURE 1. *DIABLO's* character statistics screen.

rather than the illusion of freedom offered by branching narratives and huge worlds to explore.

What I'm getting at is that a linear story must have two characteristics. As the creator of a linear RPG, you must offer the player flexibility within episodes or narrative segments or on a single map or within a single mission. Combine this flexibility with a focus on something other than narrative (such as character development) as the driving principle behind your game, and players won't notice that they're on rails, narratively speaking.

FINAL FANTASY VII does a wonderful job of allowing you to explore each of its locations with some degree of freedom. Players rarely feel constrained or stuck to a path, even though they are. The reason lies in the game's emphasis on character development. The designers recognized that freedom of movement would eventually interfere with the advancing plot, so they emphasized systems that allow players to create unique alter egos who respond to scripted events in ways that are often within the player's control. This feature allowed them to tell a better story with more interesting characters than would be present in a nonlinear game. I'm not saying that FINAL FANTASY VII

they're in control, it's as good as if they are.

Character Differentiation & Development

RPGs are character-driven. Unlike any other game genre, they rely on differentiated player characters. As such, unique, personal character growth is vital. Players must feel that they control the destiny of their alter egos and that their choices throughout the game result in increasing stature and a growing ability to impact the game world and its denizens.



Every design decision you make when crafting an RPG should first be filtered through the following simple screens:

- Does each game system, design philosophy, or mission help the character play his or her role more effectively?
- Does each serve to differentiate one character from another?

If we as game designers allow each player's character to be unique, and thus differentiate each player's experience of the game, we have been successful. To illustrate how important the

is necessarily a better game than DAGGERFALL (a non-linear game if there ever was one) — just that the designers clearly thought through the implications of the critical design decision to tell a linear story.

Here, as in most design decisions, there's no right or wrong answer. Linear narratives, expertly implemented, are no better or worse than branching narratives implemented equally well. However, it's worth pointing out that *perceived* freedom is more important than actual freedom.

If the players thinks

need to play a role is in role-playing games, and how controversial the subject can be, let me describe some personal experiences. In recent months, I found myself embroiled in a controversy that I never could have imagined. The issue involved the nature of role-playing and character identification.

It occurred at Ion Storm, where I'm currently working on the game DEUS EX. My development team, which is fairly united on role-playing design issues, suddenly found itself on the brink of civil war over whether players should be allowed to name their characters. My original plan had been to give the character a name and a back-story to go along with it. That would allow us to give the character significant relationships and, perhaps most important, a voice.

Half of the team felt that the predetermined name and identity offered too many dramatic advantages to pass up, particularly nowadays when full speech is expected and voice synthesis technology is still in its infancy. The other half of the team was appalled. "If you can't name your character," said one developer, "you're not making an RPG at all. You're making an adventure game." Several people commented that they find it annoying when they are forced to do or say things because the designer thinks their character would do or say that thing. To cut short this debate, I came up with a solution that, I believe, satisfied both camps. (You can tell me how successful my solution was when the game ships!) In any event, this argument about character names shows just how critical player identification with his or her character can be to the success of an RPG.

Statistics, Skills and/or Trackable Abilities

As tools, names are useful but not critical. In contrast, there are two core character identification and development tools: statistically driven and experientially driven story and world building. Regardless of whether you allow players to develop their characters through statistics or direct experience (or some combination of the two), you have to take a stance on the subject before you put the last period on your design document.

In games such as *FALLOUT*, *DIABLO* (Figure 1), *DAGGERFALL*, *FINAL FANTASY VII*, and *MIGHT & MAGIC VI*, numbers typically define your character's fundamental attributes (such as strength, dexterity, intelligence, and luck) and/or your character's level of accomplishment in a set of skills (such as lock picking, marksmanship, and first aid). When players run into a game problem or obstacle, they use one or more of these attributes or skills, resulting in behind-the-scenes die rolls that determine success or failure in overcoming the problem.

Development (increases in individual statistics) often comes through the expenditure of abstract skill points given by the designer for solving individual problems or for solving enough such problems to go up an arbitrary level (the rewards model). In other games, development comes through the actual use of specific capabilities in game situations (the practice model). However implemented, statistics are terrific tools for setting one character apart from another — there's a reason they've been a staple of role-playing since the birth of *D&D*.

Why are numeric systems terrific tools? For one thing, they're instantly parsable by normal human beings. Any player can tell immediately that first level isn't as good as second level, and that a strength score of 65 is better than a score of 37. In addition, die-rolls introduce tension, suspense, and variety into computer RPGs.

Statistical systems do have associated costs, though. The one that I find most damning, if only because a thoughtful designer can so easily avoid the problem, is that typically, by game's end, characters tend to end up looking more alike than different. But the trick to avoiding this is simply to impose limits on the number of skills players can select and/or to limit the number of reward points we hand out.

But using statistics poses other problems,



FIGURE 2. *FALLOUT*'s character statistics screen.

too. As easy as it is to say two characters are different — and as easy as it is to indicate these differences on a character description screen — it is extremely difficult to communicate to players the reasons why they succeeded or failed at a given task. Can players ever really know why they succeeded or failed when behind the scenes die rolls determine success of failure? Can we make players feel their contributions to character accomplishment are significant? If you choose a statistical approach, you need to provide obvious and immediate feedback when a statistic affects problem resolution.

Games such as *WASTELAND* and *DIABLO*, though separated by many years, are extremely good examples of games that use statistics and skills effectively. In these games, statistics, in addition to being rewards, allow players to refine their characters with a great deal of control and precision and to individualize their play experience in ways the games' limited storylines don't offer. *FALLOUT* is a more recent example of effective use of statistics to differentiate characters from one another (Figure 2). But

FALLOUT takes the idea of tailoring experience through statistics even further than *WASTELAND* and *DIABLO* — a player who puts his or her points into stealth and communication skills, for example, is likely to solve game problems very differently than one who puts those points into weapons skills.

Looking at most games that use the statistical character definition approach, you quickly get the impression that designers like to track numerous character statistics, and like to track them to a fine degree. Logically, this seems to be the best way to differentiate one character from another. However, if you're going to track statistics based on skill values, adjustments to these values have to be meaningful enough so that changing them makes obvious changes in the game play. Hand out too many statistical improvement rewards and characters start to look more alike than different. Create enough statistics and skills and players quickly figure out which ones matter and which don't, causing characters to look more similar. My advice is to be appropriately and thoughtfully stingy with rewards and with the number and types of statistics you provide.

But are all of these statistics really necessary? Of course not. There's another way. In recent years, a small, very vocal and extremely persuasive



FIGURE 3. *DIABLO's character inventory screen.*

minority of the design community has begun to argue in favor of statistics-free RPGs or, as some call it, the immersive experience. They feel that hidden die rolls and finely tracked statistics are unnecessary hold-overs from paper gaming. These designers pose a number of interesting questions. Why use a crutch from another medium, one with limited simulation capabilities, in computer gaming, which has far more powerful simulation tools available? Why not let player choices determine character differences? Does anyone think the difference between a 17 and an 18 in strength or between an 89 and a 90 in lock picking should have an impact on game play?

So what do these statistics-foes offer as an alternative in terms of character differentiation and the player's ability to impact a story? The two most important alternative tools available are Inventory and Skills/Special Abilities.

Most, if not all, RPGs support the accumulation and use of items by players. In most, you can pick up anything that isn't specifically nailed down and use it later, possibly even in ways the designers never imagined. In extreme cases, you end up with characters hauling around useless candy wrappers and soda cans. In the end, problem resolution is what RPGs are all about. The more tools you give the player (useless

items notwithstanding), the more solutions are likely to suggest themselves, as long as your simulation is robust enough — or your designers clever enough — to support them.

In addition to their use in problem solving, objects and weapons can be powerful tools for character differentiation. If you load up your inventory with weapons and I load mine up with keys, lockpicks, and invisibility suits, our characters look, feel, and, of necessity, behave differently from one another.

The key to making inventory a character differentiation tool is to limit, in some way, the number of items that a character can carry. In a statistics-based system, this can be accomplished by giving items weight and then tying inventory capacity to strength — how much weight a character can carry therefore becomes the limiting factor. In a statistics-free system, the same goal can be accomplished by giving each item a size and then limiting the number of things a character can lug around. Clearly, combinations of these ideas work, too — witness *DIABLO* (Figure 3) — and there are undoubtedly several other viable schemes.

For inventory to work as a character and experience differentiation tool, we must find ways to force players to make choices. Which implies limiting characters' capabilities. We as designers must be disciplined enough to parcel out items of increasing power — things that make characters more effective — in a careful, well thought-out manner. If you dumped an infinite variety of weapons in front of a character, most players would grab the most powerful one,

making the inventory limitation moot. Item and weapon differentiation must be thought of in terms of economy.

In a real-world economy, more isn't always better. The same is true in gaming. Just because you can offer players 4,000 weapons doesn't mean that you should. The choice should have meaning. Ask yourself whether you can really differentiate two weapons; if the two items offer no legitimate, significant, and obvious game play difference in your game world, why bother? When pondering inventory issues, think of yourself as the game design equivalent of Alan Greenspan of the Federal Reserve Board — you have to open and close the object floodgates to match players' capabilities to the tasks at hand. Release too much "item power" and tasks become too easy. Interest in the game wanes. Release too little "item power" and tasks become too hard. Frustration sets in. Either way, players stop playing.

Not all systems of skills and abilities depend on statistical resolution; there are plenty of other ways to structure a system. In *DEUS EX*, we use a binary action resolution scheme in which your skill level, tracked in a gross fashion (rather than a granular one) is compared to the difficulty of the task at hand. If your skill level is higher than the difficulty factor — which, in most cases, you'll know before you attempt a task — then you succeed. If your skill level is lower than the task's difficulty factor, you have to find another solution.

In a deeply simulated world (or even a modestly simulated one, such as *DEUS EX*) each game problem should be solvable in a variety of ways. In such a

simulation, a skill system like our binary action resolution model makes perfect sense. If you're

not a good enough lock picker to open a vault, maybe you're a master with explosives, or maybe you can charm a bank clerk into opening it for you.

If you're still thinking in terms of puzzles rather than obstacles, and if your world is two-dimensional rather than deeply simulated, stick with statistics.



So where does tension come into the picture? To offer levels of suspense like statistics-based systems, a statistics-free game must emphasize consequence and reward. Here's one way it can work. First, players may know in advance the outcome of a specific action (whether they do or not is largely a result of how much attention they've been paying — the designers aren't trying to hide anything). Second, players should have a fair degree of certainty about the reward for acting in a particular manner (for example, a player should know that if he or she gets through the vault door, he or she will get a million dollars). Most critically, players must not be able to do more than make an educated guess at the consequences of acting in a specific manner (for example, picking the lock on the vault door might or might not set off an alarm, blowing the door off it's hinges might attract the attention of the night watchmen or destroy the money in the vault, and charming a bank teller might allow the teller to identify you when the police show up). All actions — all choices — must reward players and, equally important, all must have consequences. There can be no right and wrong, no better and worse.

Whether statistics or experience is used to differentiate characters, some tangible measurement of character prowess and progress is necessary if a game is to be considered an RPG. In RPGs, arbitrary limitations are often placed on what your character can and cannot do. The idea of defining your character's abilities statistically is just one such arbitrary limitation.

Another is the notion of the character class. In the past, distinguishing characters using character classes has prevented mages from wearing armor, impaired warriors' use of magic, restricted clerics' use of offensive magic, and let thieves move quietly and do double damage from behind. This is the brute-force solution to the problem of differentiating characters, applicable in either statistics- or experience-based games (though fitting more easily into the former). Character classes tell players, "Here's a problem. Your character can't solve this problem in ways X, Y, and Z because those methods aren't appropriate to your class. Find another solution that takes advan-

tage of your character class's unique and clearly defined capabilities."

Character classes are not a bad way to ensure experiential differences, but they're a little inelegant. I view them as a form of "remedial role-playing"; character classes have been a crutch for novice role-players since the 1970s. If reaching the mass market is your goal, character classes could be appropriate. If you're going for the *Dungeons & Dragons* audience, character classes are a necessity. I've noticed that the use of character classes is waning, but if the upcoming title *BALDUR'S GATE* proves as successful as most industry watchers expect, it could revitalize the notion of class distinctions in role-playing.

Varied Interaction

One of the defining characteristics of role-playing is the player's ability to impact the outcome of the story through his or her actions during the game. One of the most powerful and effective ways to give players this power is to offer and reward a variety of interaction styles.

When you play an *ULTIMA*, or even a hack-and-slash game such as *DIABLO*, you meet computer-controlled people and creatures — NPCs — who don't necessarily want to kill you. In a console game such as *SUIKODEN*, you build a base of operations and forge alliances with the NPCs you choose. Together, these activities have a dramatic impact. In almost any RPG you can name, players can kill, talk to, buy or sell from, maybe even learn from NPCs. The interaction can come in the form of one-sided info-dumps or in complex, branching tree conversations. Whatever form it takes, some nonviolent interaction is a necessary, defining characteristic of RPGs.

Furthermore, you can interact with the game environment in

ways other than shooting weapons and opening doors. This may be as limited as picking up objects and manipulating switches and levers or it may be as limitless as, well, interactions in the real world. (I can dream, can't I?)

Finally, in the best RPGs, obstacles aren't limited to monsters or arbitrary puzzles. The solution to a particular game situation isn't as predictable as in more focused game categories and, often, more than one solution exists for each problem that you confront.

Combat

The effectiveness of your combat system depends on your understanding of the concept of "economy." Giving the player a big gun with unlimited ammo and then throwing a million hideous monsters at him or her isn't going to buy you much sweat and adrenaline. However, try giving the player an ordinary gun with three shots and then send two villains at him or her and see what happens.

Make sure each weapon and each enemy is radically different than all others. There's no point offering a 1911 Colt, a Glock, and a Browning Hi-Power unless there's some major game play difference between them. Hey, they're all automatic pistols and serve about the same purpose in the real world. In your game, players won't know that the stocks feel different, the



FIGURE 4. *ULTIMA VI*'S BRANCHING tree dialogue.

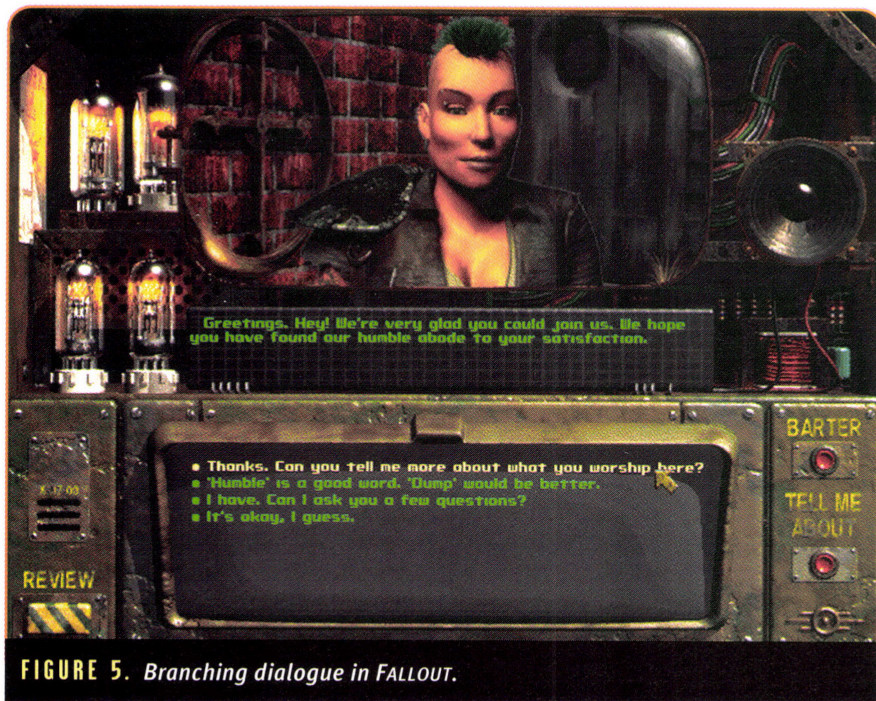
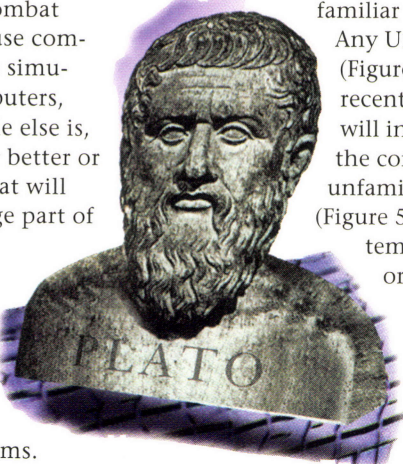


FIGURE 5. Branching dialogue in *FALLOUT*.

weight is different, and so on. And tiny variations in accuracy and kickback and the like probably won't be noticeable even if you bother trying to simulate them. Don't bother. Just put one of them in your game and be done with it. If your game fiction can support it, make certain weapons particularly useful against certain enemies. Weapon differences must be obvious and instantly apparent.

Wherever possible, differentiate your enemies as much as (or more than) you do your player characters. If, again, your game fiction supports it, give each enemy an attack that has a specific effect (or effects) on the player's ability to move, see, hear, or otherwise interact with the game.

Do these, and you have a winning combat recipe. Because combat is easy to simulate on computers, and very little else is, I suspect (for better or worse) combat will remain a large part of RPGs, and much of our design effort will continue to go into crafting new combat systems.



Conversing with NPCs

No one has yet devised and/or implemented an artful, compelling, interesting, or believable conversation system in a computer RPG. That includes everything I've done and everything you've done. No one has come up with a system that doesn't draw you out of the game world and remind you that you're just manipulating pixels on a screen. In the absence of anything better, let's look at some of the approaches that we've tried in the past.

First, there are branching-tree/key-word systems. If you've played just about any computer RPG of the last 15 years, you're familiar with these.

Any *ULTIMA* game (Figure 4) and, more recently, *FALLOUT* will introduce you to the concept, if you're unfamiliar with it (Figure 5). In this system, players read or listen to a bit of dialogue "spoken" by an NPC and are then offered a number of

response options (or are given the opportunity to type in whatever they want). Picking one of these options or typing in a likely keyword sends the NPC into another speech. Making a selection typically prevents the player from getting the information he or she would have gotten by picking another of the available response options. Eventually, the NPC runs out of things to say along a particular branch and the conversation ends, leaving the player either to start the whole conversation over and make different response option choices in an attempt to elicit additional information from the NPC, or to go talk to someone else.

The problem is that clicking through a bunch of conversation options doesn't feel much like a conversation — an interrogation, perhaps, but not a conversation. Additionally, keywords and branching trees turn the conversations themselves into puzzles. Can you guess which branch the designer wanted you to go down? The opportunity and, more often, the necessity of talking to each NPC multiple times to be sure you ferreted out the critical nugget of information or set the one necessary conversation flag is a pain and drains conversations of their emotional impact.

Another way to handle NPC interaction is through linear conversations. This is sometimes called the "NPC as signpost" approach to conversation. It's most commonly used in console RPGs, where input options are limited and storage space for branching conversations is at a premium. Basically, this



FIGURE 6. Binary decision point conversation in *SUIKODEN*.

method boils down to walking up to an NPC and having them tell you something and that's it. No interaction. Talk to them again and, unless the game state has been advanced somehow and/or the designer is particularly sharp, the NPC will simply repeat what he or she said the last time.

Linear conversations typically point you to your next goal, but they can do much more. In the best examples, NPCs can tell you about themselves and their lives. They can describe in convincing terms how you know them and how they feel about you. It's possible to evoke real emotions in a linear conversation, and about all the writer has to worry about is the role the speaker plays in the story.

A third communication solution is the use of binary decision points. This is a compromise between branching and linear conversation approaches. Most often seen in console games such as SUKODEN (Figure 6) and just about anything developed by Square, binary decision point conversations are linear except where a yes/no decision (and associated branch) will reveal something about the character, the player, or the NPC speaking. I think this is a most promising approach.

A fourth communication method is reaction-based conversations. A few designers over the years have tried a system in which NPCs speak and the player gets to pick the tone of his or her response but not the specific content (wording) of the response. This doesn't seem to offer much advantage over other, more popular systems, but it is an option.

The last communication option is simply denial. Back when Doug Church and I first started talking about SYSTEM SHOCK, we were dissatisfied with the conversation approach taken in UNDERWORLD, traditional and conventional though it may have been. And though it pained us to admit it, even to ourselves, we had no idea how to do any better. So the team designed around the unsolvable problem — we killed everyone off. The inhabitants of Citadel station would exist, for the player, only through e-mail and video logs. It was an elegant solution to an intractable problem: if we can't make you believe you're talking to a real human being, we just won't have any in our game world. (In retrospect, I

think we may have gone a little overboard — it was the right decision for that game at that time, but we failed to take into account the power of consistency and convention.)

Perhaps the best thing that can be said about conversation in computer gaming is that players have grown accustomed to inelegant, unrealistic, basically unbelievable systems and cardboard cut-out NPCs. Until someone comes up with something better, you can always fall back on convention, a fact that Doug and the SYSTEM SHOCK team didn't consider very seriously. Players "get" branching-tree/keyword systems — they're so familiar with them that they don't even think about them much anymore. And that's about the best that you can hope for — that conversation won't drag players out of your carefully crafted alternate world too badly. I await the day when voice recognition, natural language processors, basic knowledge databases, and speech synthesis become realistic options.

Exploration vs. Action

Having dealt with combat and conversation, there's only one defining characteristic of role-playing left to discuss: exploration. In the typical RPG, you can fully explore a huge, contiguous world in real time. Every ULTIMA, both UNDERWORLDS, DAGGERFALL, and countless other games have taken this approach. Uncovering all of a world's secrets is fun.

Clearly, the exploration model works in RPGs, but it's both a blessing and a curse. It's great to be able to say it takes a player hours to traverse your game's world (and that's if you don't stop

to interact with anything), but that begs the question of whether it's any fun to walk around for ten hours. Additionally, game players are increasingly pressed for time or anxious to finish one game so they can move on to the next. As a result, many seem to want smaller worlds, shorter play times, and more frequent pats on the back. I know there are still hardcore game players out there who always want more, but you must decide whether there are enough of these consumers to support your development budget.

Recent trends favor smaller, deeply simulated worlds over large, contiguous spaces an inch deep and miles wide, at least in terms of depth of simulation. Further, many RPGs these days — my own DEUS EX, THIEF: THE DARK PROJECT, DIABLO, and others — are adopting a mission orientation. They break the world up into more manageable sections to minimize walking



FIGURE 7. ULTIMA VII uses a top-down, third-person perspective.



FIGURE 8. Standard game play window in MIGHT & MAGIC VI, showing the first-person gaming experience.

around and maximize fun. Mission structure also goes hand-in-hand with linearity and, together, they allow us to tell the best stories possible.

Variables

Now that we've discussed the defining characteristics — the rules of role-playing, if you will — what of the variables?

There are certainly characteristics of role-playing that are not universal, that you can adopt or ignore, as you wish. Here are some of them.

CAMERA POSITION. I'm the last person qualified to address technology, but camera positioning is an important enough issue to cover briefly. There are several ways to approach camera positioning and player point of view in role-playing. The most common perspectives are: first-person, three-quarters-overhead third-person (as in, an isometric viewpoint), and top-down third-person (Figure 7). For the purposes of this discussion, and for the sake of brevity, I'll treat the third-person perspectives as the same.

In a genre that, at some level, boils down to providing the player with the ultimate "I did this" experience, what could be more compelling than entering a new world and seeing it through your own eyes? The *UNDERWORLD* and *MIGHT & MAGIC* series (Figure 8), not to mention many others, offer fine first-person role-playing experiences. If what you're after is simulating a world, a first-person view goes a long way toward making the player feel as though he or she is "there." You're reducing the distance between player and character to almost nothing.

One drawback to the first-person perspective is that it puts players at a tactical disadvantage by limiting their awareness of what's going on behind and to the sides of their characters. And if you're committed to turn-based, tactically challenging combat, convoluted conversations that take place on a separate conversation screen, and the video equivalent of

paper game character sheets crawling with attributes, skills, and numbers, first-person could be the worst choice for your game.

If you're trying to recreate the paper gaming experience or the experience you get when you read a great novel or watch a film — the fiction experience — a third-person view may be just what you need. To capture that "fiction feel," it's good to let players guide their characters rather than *be* their characters. You want a bit of distance between player and character.

A third-person perspective is also ideal for tactical decision-making, particularly when combined with a turn-based combat system, a character sheet, and a separate conversation screen. Each of these elements contributes to the player's ability to make informed decisions about how to develop his or her character. The trade-off is that it's tough to care much about NPCs that are obviously nothing more than bunches of pixels an inch high when you're looking at them from a bird's perspective.

LONE ADVENTURER OR PARTY? If you're working on an RPG, there's one question I guarantee you've been asked: "Is there gonna be multiplayer support?" Everybody seems to want to go exploring with a party. In a paper game, where players gather around a table, engage in collective acts of imagination, and push lead miniatures representing their characters around on tabletops, the party idea works just fine. Certainly, a case could be made that allowing a party of adventurers to go through your story together, linked via modem, LAN, or the Internet, is a worthy pursuit despite the problems of communication and coordination among party members.

But what of the single-player RPG? Does the party make sense in that context? Many classic RPGs indicate that it does. The early *ULTIMAS* (among many others) allow a single player to take control of a party of adventurers. However, I think controlling parties in single-player, story-based RPG is a bad idea, particularly when it's a real-time game. There are several reasons I feel this way.

First, if one of the primary goals of role-playing is to allow players to cre-

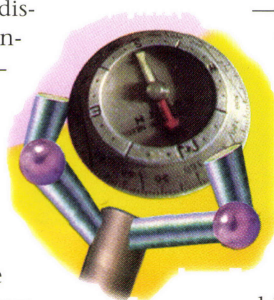
ate an alter ego, you should do everything possible to increase a player's identification with his or her character. When controlling a party, player identification with a single character is history — as is role-playing, in my opinion. At that point, you're playing a boardgame. Second, AI limitations mean you're inevitably going to be slowed down by teammates who can't think quickly on their feet and, even slowly, can't respond the way real people would.

If you subscribe to the ideal that players should describe their adventures by saying "I did" something rather than "Lara Croft did" something or the "The Avatar did" something, controlling a party is a problem. In recent years, more and more designers seem to be coming around to this mode of thinking. (Recent *ULTIMAS*, *UNDERWORLD*, *DIABLO*, and *DAGGERFALL* have all adopted the solo model, often to the chagrin of fans.) Solo play is simpler to implement, speeds up game play, and fosters a direct connection between player and character that seems critical to RPG success.

If you choose to include full-party control in your single-player game, recognize that you'll reduce player involvement and turn off people who value tactical thinking in games, and use it in conjunction with turn-based combat and a third-person perspective. A real-time, first-person party-based game is just asking for trouble.

It's Time to Advance the Genre

Computer role-playing games are no longer just an infant medium learning to crawl. They've been around for a while, and now we, their designers, must start figuring out what we are, and what we can do to make the most of them. I've got no beef with folks who want to continue recreating the past. Just don't count me among the people who think that's good enough. Our goal should be to create games that are simple to learn and play, accessible to the broadest possible audience, and yet offer enough depth that hardcore game players will flock to them. ■



[milia'99]

February

9-12

There's this guy who claims that only a couple of years ago, he was just another developer in the middle of nowhere and that, thanks to just one trip he made to the French Riviera where he exhibited at this high profile interactive media trade show in Cannes, his company has gone public and he's now got offices in London, Tokyo, Los Angeles, and Amsterdam. He also goes on and on about the show and how he rubbed shoulders with the digerati elite on the showfloor, at the conferences and parties, and the famous Martinez bar. Plus, he says the show was just overflowing with journalists from all over the world... You'll probably think he's dreaming.

But he's not dreaming.

The show really does exist and it's called Milia.

Milia's the world's most effective and innovative content exhibition and conference for interactive media & entertainment professionals. It's the only show that connects the entire chain of on and off-line content development from concept to distribution.

... And it really is on the French Riviera.

THE INTERNATIONAL CONTENT MARKET FOR INTERACTIVE MEDIA

PALAIS DES FESTIVALS - CANNES - FRANCE

Just call Patrick Lynch to find out about it.
Tel : (212) 689 42 20 - Fax : (212) 689 43 48

Implementing Coordinated Movement

by Dave C. Pottinger

48



art of the fun of working in the game industry is

the constant demand for technical innovations

that will allow designers to create better games.

In the real-time strategy (RTS) genre, most

developers are focusing on improving group

movement for their next round of games. I'm not

talking about the relatively low-tech methods cur-

rently in use. Instead, I'm referring to coordinated group movement, where units cooperate with each other to move around the map with intelligence and cohesion. Any RTS game developer that wants to be competitive needs to look beyond simple unit movement; only the games that weigh in with solid coordinated movement systems will go the distance.

In this article, the second and final part of my coordinated unit movement series, we'll take a look at how to use the systems that we considered in the first article to satisfy our coordinated group movement goal. We'll also examine how we can use our coordinated movement fundamentals

to solve some classic, complex movement problems. While we will spend most of our time talking about these features through the RTS microscope, they can easily be applied to other types of games.

Catching Up

Last month, we discussed a lot of the low-level issues of coordinated unit movement. While pathfinding (the act of finding a route from point A to point B) gets all of the press, the movement code (the execution of a unit along a path) is just as important in creating a positive game experience.

A game can have terrific pathfinding that never fails to find the optimum path. But, if the movement system isn't up to par, the overall appearance to the players is going to be that the units are stupid and can't figure out where to go.

One of the key components to any good movement system is the collision determination system. The collision system really just needs to provide accurate information about when and where units will collide. A more advanced collision system will be continuous rather than discrete. Most games scale the length of the unit movement based on the length of the game update loop. As the length of that update loop increases, the gap between point A and point B can get pretty large. Discrete collision systems ignore that gap, whereas continuous systems check the gap to make sure there isn't anything in between the two points that would have created a collision with the unit being moved.

After several close calls, Dave managed to avoid getting a "real job" and joined Ensemble Studios straight out of college a few years ago (just in time to do the computer player AI for a little game called AGE OF EMPIRES). These days, Dave spends his time either leading the development of Ensemble Studios' engines or with his lovely wife Kristen. Dave can be reached at dpottinger@ensemblestudios.com.

Continuous collision determination systems are more accurate and more realistic. They're more difficult to write, though.

Another important element for coordinated unit movement is position prediction. We need to know where our units are trying to go so that we can make intelligent decisions about how to avoid collisions. Although building a fast position-prediction system presents us with a number of issues, for this article, we can assume that our collision determination system has been augmented to tell us about future collisions in addition to current collisions. Thus, each unit in the game will know with which units it's currently in collision with and which units it will collide with in the near future. We presented several rules for getting two units out of collision in last month's article.

All of these elements work together to create the basis for a solid, first-order (single unit to single unit) coordinated movement system. The core thing to keep in mind for this article is that we have an accurate, continuous collision determination system that tells us when and where units will collide. We'll use that collision system in conjunction with the collision resolution rules to create second order (three or more units/groups in collision) coordination.

Group Movement

Looking at the definition of a group (see the sidebar, "Units, Groups, and Formations"), we can immediately see that we need to store several pieces of data. We need a list of the units that make up our group, and we need the maximum speed at which the group can move while still keeping together. Additionally, we probably want to store the centroid of the group, which will give us a handy reference point for the group. We also want to store a commander for the group. For most games, it doesn't matter how the commander is selected; it's just important to have one.

One basic question needs to be answered before we proceed, though. Do we need to keep the units together as they move across the board? If not, then the group is just a user interface

convenience. Each unit will path and move as if the player had issued individual commands to each group member. As we look at how to improve on the organization of our groups, we can

see that there are varying degrees of group movement cohesion.

UNITS IN A GROUP JUST MOVE AT THE SAME SPEED. Usually, this sort of organization moves the group at the maximum

Units, Groups, and Formations

Unit. A game entity that has the ability to move around the map. Players expect their units to act intelligently.

Group. A general collection of units that have been grouped together by the user for convenience (usually to issue the same order to all of the units in the group). Other than a desire to keep its units together, groups don't place any other restrictions on unit movement.

Formation. A more complex group. A formation has an orientation (a front, a back, a right flank, and a left flank). Each unit in the formation tries to maintain a unique relative position within the formation. More complex models provide an individualized unit facing within the overall formation and support for wheeling during movement.

FIGURE 1. All grouped units should be kept together.

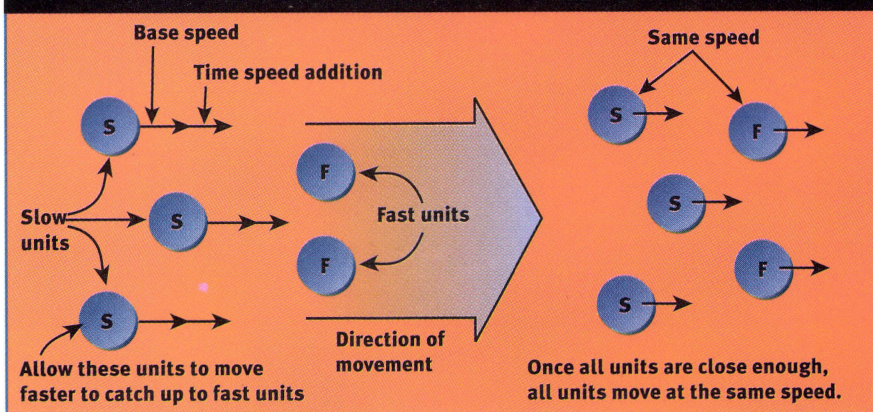
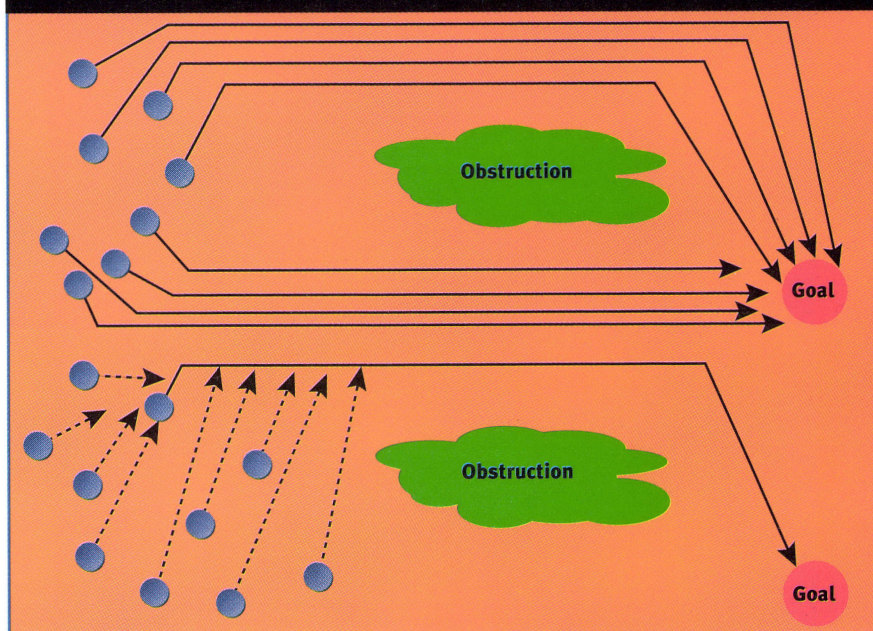


FIGURE 2. Units in a group should follow the same path.



speed of its slowest unit, but sometimes it's better to let a slow unit move a little faster when it's in a group (Figure 1). Designers generally give units a slow movement speed for a reason, though; altering that speed can often create unbalanced game play by allowing a powerful unit to move around the map too quickly.

UNITS IN A GROUP MOVE AT THE SAME SPEED

AND TAKE THE SAME PATH. This sort of organization prevents half of the group's units from walking one way around the forest while the other half takes a completely different route (Figure 2). Later, we'll look at an easy way to implement this sort of organization.

UNITS IN A GROUP MOVE AT THE SAME SPEED, TAKE THE SAME PATH, AND ARRIVE AT THE SAME TIME. This organization exhibits the

most complex behavior that we'll apply to our group definition. In addition to combining the previous two options, it also requires that units farther ahead wait for other units to catch up and possibly allows slower units to get a temporary speed boost in order to catch up.

So, how can we achieve the last option? By implementing a hierarchical movement system, we can manage individual unit movement in a way that allows us to consider a group of units together. If we group units together to create a group object, we can store all of the necessary data, calculate the maximum speed for the group as a whole, and provide the basic decision making regarding when units will wait for other units (Listing 1).

The `BGroup` class manages the unit interactions within itself. At any point in time, it should be able to develop a schedule for resolving collisions between its units. It needs to be able to control or modify the unit movement through the use of parameter settings and priority manipulation. If your unit system only has support for one movement priority, you'll want to track a secondary movement priority within the group for each unit in the group. Thus, to the outside world, the group can behave as a single entity with a single movement priority, but still have an internal prioritization. Essentially, the `BGroup` class is another complete, closed movement system.

The commander of the group is the unit that will be doing the pathfinding for the group. The commander will decide which route the group as a whole will take. For basic group movement, this may not mean much more than the commander being the object that generates the pathfinding call. As we'll see in the next section, though, there's a lot more that the commander can do.

LISTING 1. The `BUnitGroup` class.

```

//*****
// BUnitGroup
//*****
class BUnitGroup
{
public:
    BUnitGroup( void );
    ~BUnitGroup( void );

    //Returns the ID for this group instance.
    int          getID( void ) const { return(mID); }

    //Various get and set functions. Type designates the type of the group
    //(and is thus game specific). Centroid, maxSpeed, and commander are
    //obvious. FormationID is the id lookup for any formation attached to
    //the group (will be some sentinel value if not set).
    int          getType( void ) const { return(mType); }
    void         setType( int v ) { mType=v; }
    BVector&     getCentroid( void ) const { return(mCentroid); }
    float        getMaxSpeed( void ) const { return(mMaxSpeed); }
    int          getCommanderID( void ) const { return(mCommanderID); }
    BOOL         getFormationID( void ) const { return(mFormationID); }
    BOOL         setFormationID( int fID );

    //Standard update and render functions. Update generates all of the
    //decision making within the group. Render is here for graphical
    //debugging.
    BOOL         update( void );
    BOOL         render( BMatrix& viewMatrix );

    //Basic unit addition and removal functions.
    BOOL         addUnit( int unitID );
    BOOL         removeUnit( int unitID );
    int          getNumberUnits( void ) const { return(mNumberUnits); }
    int          getUnit( int index );

protected:
    int          mID;
    int          mType;
    BVector      mCentroid;
    float        mMaxSpeed;
    int          mCommanderID;
    int          mFormationID;
    int          mNumberUnits;
    BVector*     mUnitPositions;
    BVector*     mDesiredPositions;
};

```

Basic Formation Movement

Formations build on the group system. Formations are a more restrictive version of groups because we have to define a very specific position for each unit within the group. All of the units must stay together during group movement in terms of speed, path, and relative distance; it doesn't do any

good to have a column of units if there are huge gaps in that column while it's moving around the map.

The `BFormation` class (Listing 2) manages the definition of the desired positions (the positions and orientations that we want for each unit in the formation), the orientation, and the state of the formation. Most formations that a game uses are predefined; it's useful to make these easy to edit during development (via a text file or something else that a nonprogrammer can manipulate). We do want the ability to create a formation definition on the fly, though, so we'll take the memory hit and have each formation instance in the game maintain a copy of its own definition.

Under this model, we must also track the state of a formation. The state `cStateBroken` means that the formation isn't formed and isn't trying to form. `cStateForming` signifies that our formation is trying to form up, but hasn't yet reached `cStateFormed`. Once all of our units are in their desired positions, we change the formation state to `cStateFormed`. To make the movement considerably easier, we can say that a formation can't move until it's formed.

When we're ready to use a formation, our first task is to form the formation. When given a formation, `BGroup` enforces the formation's desired positions. These positions are calculated relative to the current orientation of the formation. When the formation's orientation is rotated, then the formation's desired positions will automatically wheel in the proper direction.

To form the units into a formation, we use scheduled positioning. Each position in the formation has a scheduling value (either by simple definition or algorithmic calculation) that will prioritize the order in which units need to form. For starters, it works well to form from the inside and work outward in order to minimize collisions and formation time (Figure 3). The group code manages the forming with the algorithm shown in Listing 3.

So, now that we have all of our swordsmen in place, what do we do with them? We can start moving them around the board. We can assume that our pathfinding has found a viable path (a path that can be followed) for our formation's current size and shape (Figure 4). If we

LISTING 2. The `BFormation` class.

```

//*****
// BFormation Class
//*****
class BFormation
{
public:
    //The three formation states.
    enum
    {
        cStateBroken=0,
        cStateForming,
        cStateFormed
    };

    BFormation( void );
    ~BFormation( void );

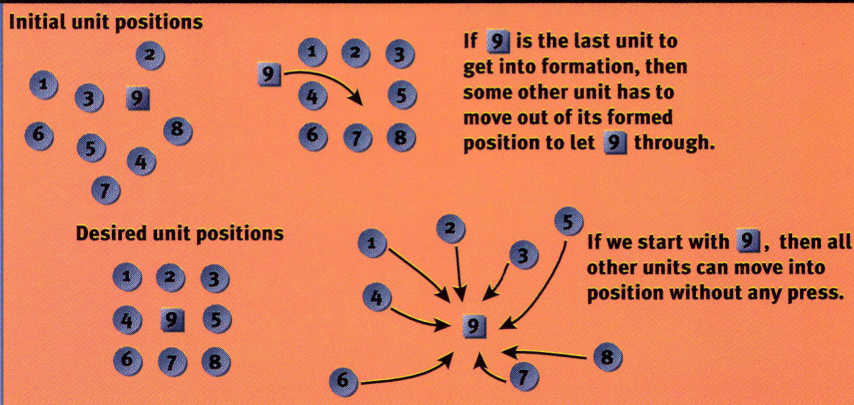
    //Accessors for the formation's orientation and state. The expectation
    //is that BFormation is really a data storage class; BGroup drives the
    //state by calling the set method as needed.
    BVector& getOrientation( void ) { return(mOrientation); }
    void setOrientation( BVector& v ) { mOrientation=v; }
    int getState( void ) const { return(mState); }
    void setState( int v ) { mState=v; }

    //The unit management functions. These all return information for the
    //canonical definition of the formation. It would probably be a good
    //idea to package the unit information into a class itself.
    BOOL setUnits( int num, BVector* pos, BVector* ori, int* types );
    int getNumberUnits( void ) const { return(mNumberUnits); }
    BVector& getUnitPosition( int index );
    BVector& getUnitOrientation( int index );
    int getUnitType( int index );

protected:
    BVector mOrientation;
    int mState;
    int mNumberUnits;
    BVector* mPositions;
    BVector* mOrientations;
    int* mTypes;
};

```

FIGURE 3. Start forming the formation from the inside out.



LISTING 3. The formation algorithm.

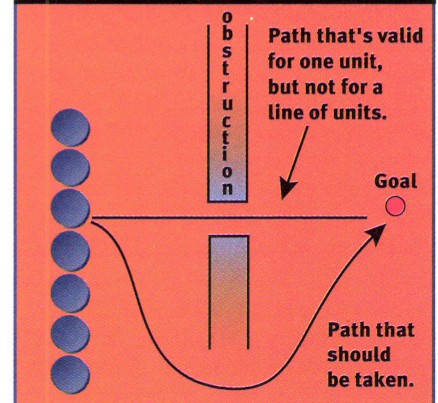
```

Set all units' internal group movement priorities to the same low priority value.
Set state to cStateForming.
While state is cStateForming:
{
    Find the unfilled position that's closest to the center of the formation.
    If no unit was available
        Set the state to cStateFormed and break out of forming loop.

    Select a unit to fill that slot using a game specific heuristic that:
        Minimizes the distance the unit has to travel.
        Will collide with the fewest number of other formation members.
        Has the lowest overall travel time.

    Set unit's movement priority to a medium priority value.
    Wait (across multiple game updates) until unit is in position.
    Set unit's movement priority to highest possible value. This ensures that
    subsequently formed units will not dislodge this unit.
}
    
```

FIGURE 4. The pathfinding needs to find a path that's usable given the formation's size and shape.



We have a couple of ways to deal with direction changes for a formation: we can ignore the direction change or we can wheel the formation to face in the new direction. Ignoring the direction change is simple and is actually appropriate for something such as a box formation (Figure 5). Wheeling isn't much more complicated and is very appropriate for something such as a line. When we want to wheel, our first step is to stop the formation from moving. After rotating the orientation of the formation, we recalculate the desired positions (Figure 6). When that's done, we just go back to the cStateForming state, which causes the group code to move our units to their new positions and sets us back to cStateFormed when it's done (at which point, we can continue to move).

FIGURE 5. Wheeling doesn't apply to every formation.

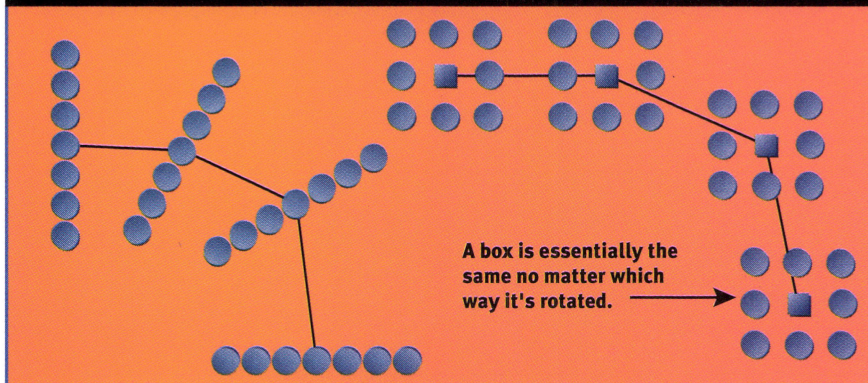


FIGURE 6. Recalculate the desired positions after the formation's orientation has changed.

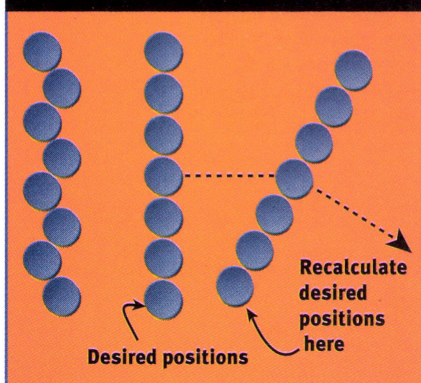
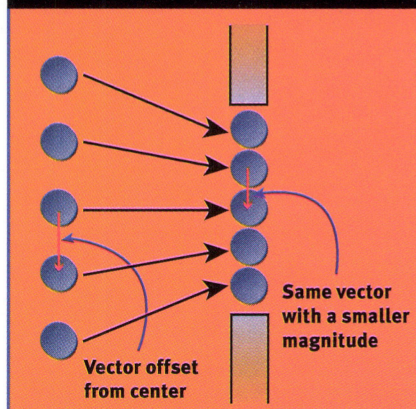


FIGURE 7. Scaling desired positions to fit through small gaps.



Advanced Formation Movement

So, now we've got formations moving around the map. Because our game map is dynamic and complex, it's possible that our planned path will be invalidated. If that happens, we'll need to manipulate the formation in one of three ways.

SCALING UNIT POSITIONS. Because the desired positions are really just vector offsets within a formation, we can apply a scaling factor to the entire formation to make it smaller. And a smaller formation can fit through small gaps in walls or treelines (Figure 7). This method works well for formations in which the units are spread out, but it's pretty useless for formations where the

Miller Freeman Game Group Research

PUBLISHER PROFILES 1999

STRATEGY, PERFORMANCE & ANALYSIS OF THE TOP GAME PUBLISHERS

The Game Industry has never been more competitive, and the landscape is changing rapidly. If you want to know what the top publishers in the interactive entertainment market are doing, the 1999 Publisher Profiles Report is a must-have addition to your strategic arsenal.

INCLUDED IN THE REPORT:

The 1999 Publisher Profiles Report includes in-depth analysis of the operational, financial, developer-relations and marketing strategies of the 40 top public & private publishers from North America, Europe, and Japan. Pre-order your copy today and receive (for each company reviewed):

- *Company overview summarizing strategic direction*
- *Business analysis and competitive stance*
- *Ownership and management structure*
- *Financial analysis and performance*
- *Developer relationships, agreements & assets*

UNSURPASSED DETAIL:

Key financial data for each company tells you which publishers are on top, and which upstarts could supplant them next year. See how the top publishers compare against each other based on revenue, profits, operating expenses, and other metrics. Take advantage of the Miller Freeman Game Group's extensive industry analysis, and get rankings of the top 40 publishers by key criteria such as:

- *Sales*
- *Operating earnings, and game-specific earnings*
- *Net income, assets, cash flow & capital spending*
- *Game development capacity*
- *Number of employees*
- *Technical strengths*
- *Return on equity, capital and assets*
- *Operating, pretax and net margins and*
- *Growth in sales & earnings*

Pre-Order your copy today to save over 10%, and reserve your copy of the 1999 MFG Research Publisher Profiles Report. Get that competitive edge at: <http://www.mfgame.com/research/> and place your order on our fast & secure website. Your copy will ship in Q1 1999.

Now through December 30th, the Publisher Profiles Report is available for only \$895 (\$995 after December 30th).

3D GRAPHICS MARKET OUTLOOK

IF THE 3D GRAPHICS MARKET IMPACTS YOUR BUSINESS, THEN YOU NEED EXPERT ANALYSIS. Omid Rahmat, Principal, Doodah Marketing, and industry consultant, has taken an in-depth look at the hottest technologies and the companies creating them. Who owns the market? Who is up and coming? Where should YOU place your development resources in 1999 and beyond?

This 30 page report will help you target and support the right hardware while cutting down your production schedule. Help avoid costly mistakes by taking advantage of this detailed report, including:

- *The factors driving consumer 3D*
- *Projected sales through 1999*
- *Movers, shakers, influencers and up-and-comers*
- *The impact of the 3D API*
- *OEMs role in Consumer 3D*
- *Company information from 3Dfx, 3DLabs, Cirrus Logic, S3, nVidia, ATI, Evans & Sutherland, Intel, Intergraph, Fujitsu and more!*

The 1999 3D Graphics Market Outlook is the only report designed specifically for the game development community, giving you exactly what you need to make great 3D games, on time and on budget.

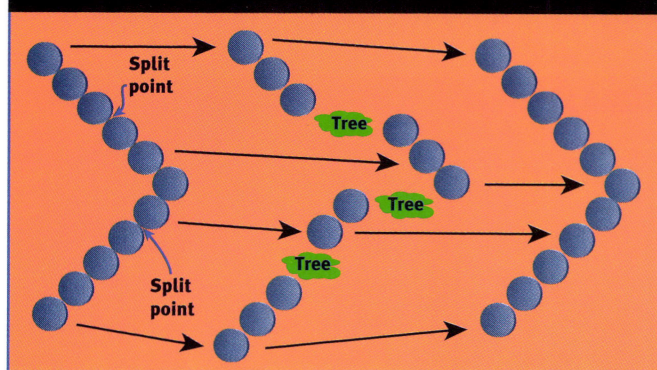
To find out more, or to pre-order your copy at the special introductory price of \$495, please see: www.mfgame.com/research/

AVAILABLE Q1 1999

To order either report,
or to get more information,
see <http://www.mfgame.com> today!

Miller Freeman Game Group

FIGURE 8. Reform the formation after the obstacles.



units are already shoulder to shoulder (as in a line). Scaling the offsets down in that case would just make our swordsmen stand on top of each other, which isn't at all what we want.

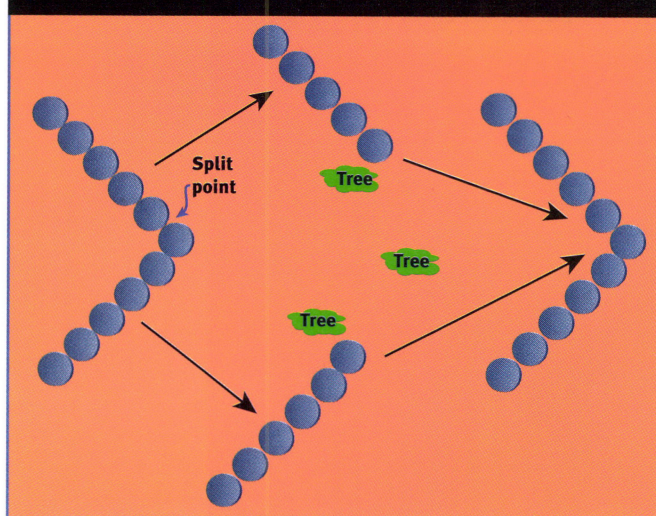
SIMPLE ORDERED OBSTACLE AVOIDANCE. If we're moving a formation and we encounter a collision with another game entity (either a current collision or a future collision), we can assume that our path is still viable, with the exception of this entity being in the way. The simple solution is to find the first place along our formation's path where it will not be in collision with the entity and reform our formation at that spot (Figure 8). Thus, the line of infantry will break, walk around the obstacle, and reform on the other side. This solution can fall apart fairly easily, though, so it's important to realize when the reformation position is too far along the path to be of use. If the distance around the obstacle is far enough that it interferes

with the reformation of your group, then you should just repath your formation.

HALVING AND REJOINING

While simple avoidance is a good solution, it does dilute the visual impact of seeing a formation move efficiently around the map. Halving can preserve the visual impact of well-formed troops. When we encounter an obstacle that's within the front projection of the formation (Figure 9), we can pick a split point and create two formations out of our single formation. These formations then move to the rejoin position and then merge back into one formation. Halving is a simple calculation that dramatically increases the visual appeal of formations.

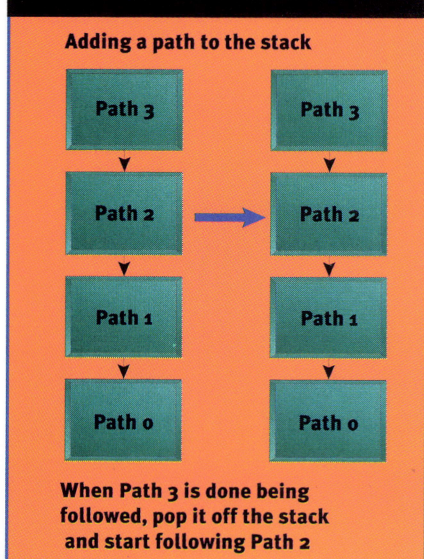
FIGURE 9. Split the formation in two to get around an obstacle.



unit is following, the current waypoint the unit is moving toward, and whether or not the unit is on a patrol. A path stack suits our needs in two significant ways.

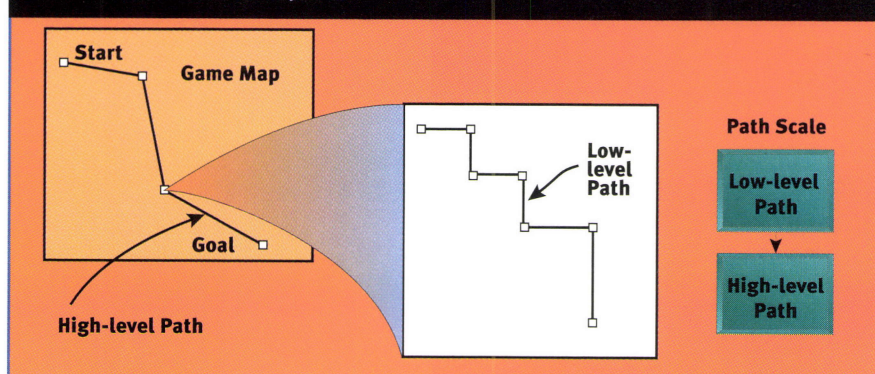
First, it facilitates a hierarchical pathfinding setup (Figure 11). Game developers are beginning to realize that there are two distinctly different types of pathfinding: high-level and low-level. A high-level path routes a unit around major terrain obstacles and chokepoints on a map, similarly to how a human player might manually set waypoints for a unit. A low-level path deals with avoidance of smaller obstacles and is much more accurate on details. A path stack is the ideal method for storing this high- and low-level information. We can find a high-level path and stuff that into the stack. When we need to find a low-level path (to avoid a future collision with that single tree in the big field), we can stuff

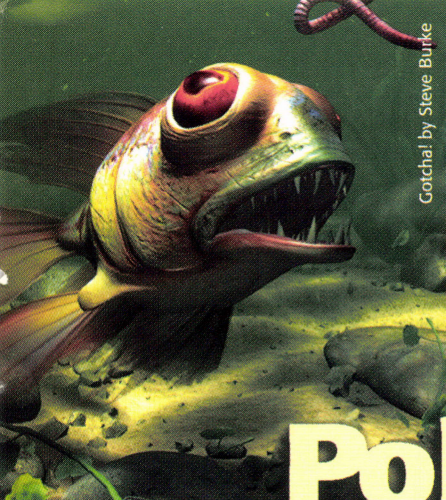
FIGURE 10. Path stack.



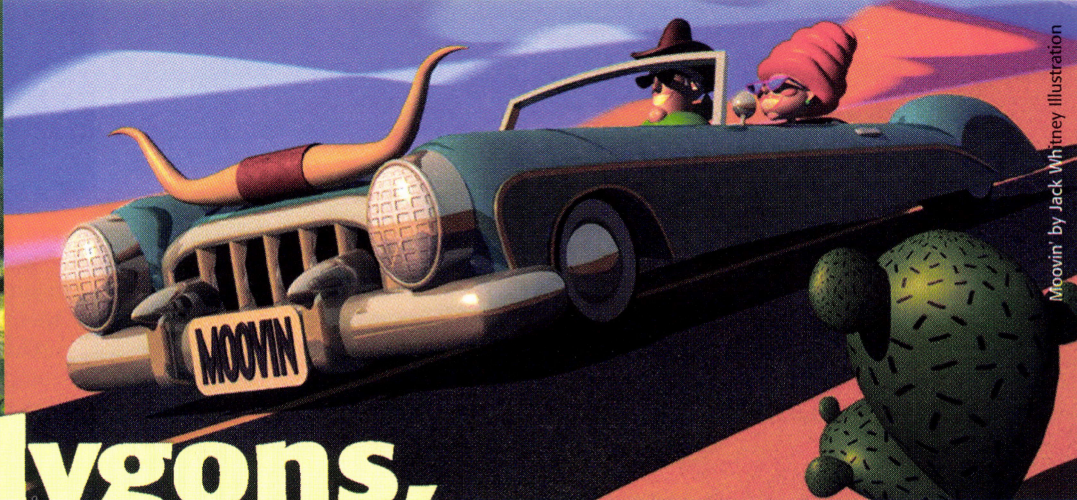
A path stack is a simple stack-based (last in, first out) method for storing the movement route information for a unit (Figure 10). The path stack tracks information such as the path the

FIGURE 11. Hierarchical paths.





Gotchal by Steve Burke



Moovin' by Jack Whitney Illustration

Polygons, NURBS, and Splines... Oh My!

"Tomorrow Never Dies" image animated with Softimage 3D by Framestore/LimeLight/Eon

Dune House, ©Tom Marlin, www.Marlinstudios.com, trueSpace 4

3D

Design & Animation

CONFERENCE & EXPOSITION

Conference:
May 10-13, 1999

Exposition:
May 11-12, 1999

**Santa Clara
Convention Center,
Santa Clara, CA**

**This is must-attend training for
advanced graphics professionals in:**

Architectural Visualization ■ Animation ■ Game Development
Broadcast & Film Design ■ Special Effects ■ Video ■ Web Design

Request a catalog today!

Yes! Send me a free copy of the 3D Design & Animation Conference & Exposition catalog.

Name _____ Title _____

Company _____

Address _____ City _____ State _____ Zip _____

Phone _____ Fax _____ Email _____

TGD0299

in Miller Freeman

Learn the Latest Technologies.

You're not in Kansas anymore. There are whole worlds you can make—if you know the tools and techniques. Create your own cast of characters and tackle the latest technologies at the 3D Design & Animation Conference & Exposition.

Choose from over 50 classes in four intense days of conference sessions taught by some of the top animators, 3D designers, and technology innovators.

At the conference, you'll recharge your creativity and take your work to a new dimension!

Topics Include:

- Animation
- Character design and development
- Special effects
- Rendering
- Modeling
- Games
- Texture mapping
- Mixing live action and CG
- Storyboarding
- Architectural visualization
- Breaking into the industry
- Web
- and much, much more!

Platinum Sponsor:

intel

Gold Sponsors:

E&S
EVANS & SUTHERLAND

3Dlabs

Media Sponsor:

3D
Design

Media Co-Sponsors:

ANIMATION
WORKSHOPS

CADENCE

Game
DEVELOPER

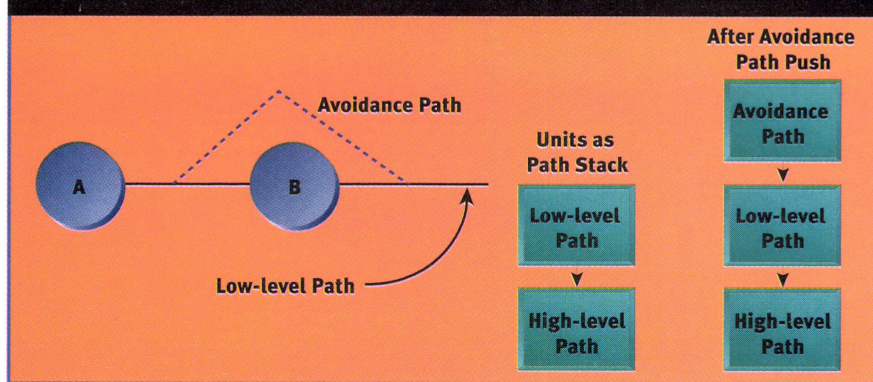
DV

Phone:
800-789-2223
(outside US phone: 817-255-8050)

Fax:
815-255-8070
(outside US fax: 817-255-8070)

Web: www.3dshow.com

FIGURE 12. Push a temporary avoidance path onto the path stack



we pass those two units into the collision resolution system.

As soon as the collision between the first two units is resolved (Figure 14), we need to reevaluate the collision and update the unit involvement. A more complex system could handle the addition of new units to the collision at this point, but you can get good results by simply removing units as they resolve their collisions with the original units. Once we've updated the units in the collision, we go back to find two more units to resolve; we repeat this process until no more units are involved in the collision (Figure 15).

You can implement this system in two different areas: the collision resolution rules or the collision determination system. The collision resolution rules would need to be changed in the way in which they units higher and lower priority; these rules aren't particularly difficult to change, but this modification does increase the complexity of that code. Alternatively, you can change the collision determination system so that it only generates collisions that involve two units at a time; you still have to find all of the units in a collision, though, before you can make this decision.

Solving the Stacked Canyon Problem

One of the ultimate goals of any movement system is to create intelligent movement. Nothing looks more intelligent than a system that solves the stacked canyon problem (Figure 16). The stacked canyon isn't a simple problem to solve upon first inspection, but we can reuse some simple scheduling to solve it once we have our coordinated unit movement in place.

The first step is to identify that you have a stacked canyon problem. This step is important because it's needed to propagate the movement priority of the driving unit (the unit trying to move through the stacked canyon) through to the rest of the units. We could just let each unit ask other units to move out of the way based on its own priority, but a better solution to use the priority of the driving unit — after all, that's the unit that we really want to get through the canyon. Identifying a stacked canyon problem

more paths onto the stack and execute those. When we're done executing a path on the stack, we pop it off the stack and continue moving along the path that's now at the top of the stack.

Second, a path stack enables high-level path reuse. If you recall, one of the key components to good group and formation movement is that all of the units take the same route around the map. If we write our path stack system so that multiple units can reference the same path, then we can easily allow units to reuse the same high-level path. A formation commander would find a high-level path and pass that path on to the rest of the units in his formation without any of them having to do any extra work.

Structuring the path storage in this manner offers us several other benefits. By breaking up a high-level path into several low-level paths, we can refine future low-level segments before we execute them. We can also delay finding a future low-level path segment if we can reasonably trust that the high-level path is viable. If we're doing highly coordinated unit movement, a path stack allows us to push temporary avoidance paths onto the stack and

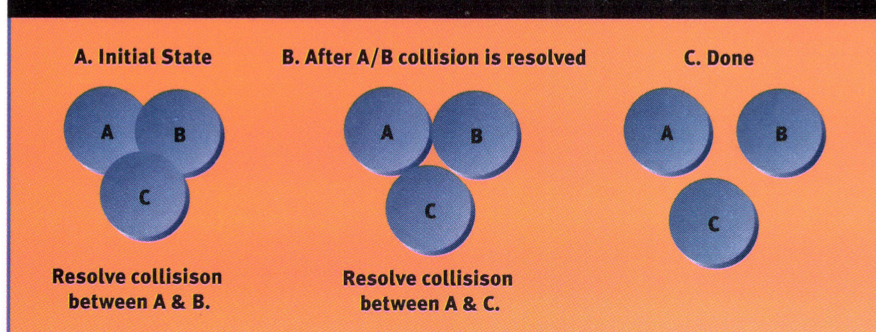
have them easily and immediately integrated into the unit's movement (Figure 12).

Solving a Compound Collision

For our purposes, compound collisions are defined as simultaneous collisions between more than two units. Most games will have a practical upper limit to how many units can be involved in a compound collision. Still, as soon as a collision involves more than two units, programmers generally end up writing a lot of spaghetti logic that breaks way too easily. But we'll avoid that situation by reusing the movement priorities and doing some simple scheduling.

If we have a compound collision between three units (Figure 13), our first task is to find the highest-priority unit involved in the collision. Once we've identified that unit, we need to look at the other units in the collision and find the most important collision for the highest priority unit to resolve (this may or may not be a collision with the next highest-priority unit in the collision). Once we have two units,

FIGURE 13. Solving a compound collision. The order of priority is A → B → C.



Create. Communicate. Innovate!



**You do it every day.
How can you do it better?**

NAB99 is your one-stop shop for ideas, innovations and solutions.

If you've never been to NAB, ask anyone who has—and you'll learn that it's more than just a trade show. NAB99 is the industry rally point where you can explore the entire convergence marketplace. Celebrate creativity. Spark new mass media potentials for the Internet. And decide what "out-of-the box" technologies and business solutions to buy, and from whom—all in one place, all at one time.

This is the power and spirit of NAB99!

Start planning today!

www.nab.org/conventions/

Or call 1.800.342.2460

or 1.202.429.4194

NEW @ NAB99!

Weekend Digital Video Production Workshops—That means no lost time on the exhibit floor!

© Digital Video Solutions

© Creating Cutting-Edge Digital Content

co-produced with **DV**

Plus, special dual panel "Super-Sessions", including The Virtualization of Hollywood—

co-produced with **Performance Animation Society**



April 17-22, 1999
Exhibits/April 19-22
Las Vegas
Nevada/USA

can be done in a couple of ways: noticing that the driving unit will push the first unit into a second unit or looking at the driving unit's future collision list to find multiple collisions. Whichever method is used, the pushed unit should move with the same priority as the driving unit.

Once we've identified the problem, we have a fairly simple recursive execution using the coordinated movement (Figure 17). We treat the first pushed unit as the driving unit for the second pushed unit, and so on. Each unit is pushed away from its driving unit until it can move to the side. When the last unit moves to the side (Figure 18), the original driving unit has a clear path by which to move through the canyon.

A nice touch is to restore the canyon units to their original states. To do this, we simply need to track the pushing and execute the moves in the reverse order from which the units moved to the side (Figure 19). It's also useful to have the movement code recognize when the driving unit is part of a group so that the rest of the group's units can move through the canyon before the canyon units resume their original positions.

Tips

OPTIMIZE THIS GENERAL SYSTEM TO YOUR GAME. A lot of extra computation can be eliminated or simplified if you're only doing a 2D game.

Regardless of whether you're doing a 2D or 3D game, your collision detection system will need a good, highly optimized object culling system; they're not just for graphics anymore.

USE DIFFERENT METHODS FOR HIGH- AND LOW-LEVEL PATHFINDING. To date, most games have used the same system for both solutions. Using a low-level solution for high-level pathfinding generally results in high-level pathfinding that's slow and not able to find long paths.

Conversely, a high-level pathfinder used for low-level pathfinding creates paths that don't take all of the obstacles into account or are forced to allow units to move completely through each other. Bite the bullet and do two separate systems.

NO MATTER WHAT YOU DO, UNITS WILL OVERLAP. Unit overlap is unavoidable or, at best, incredibly difficult to prevent in all

cases. You're better off simply writing code that can deal with the problem early. Your game will be a lot more playable throughout its development.

GAME MAPS ARE GETTING MORE AND MORE COMPLEX. Random maps are going to be one of the key discriminating features in RTS games for some time to come. The better movement systems will handle random maps and also take changing map circumstances into account.

UNDERSTAND HOW THE UPDATE AFFECTS UNIT MOVEMENT. Variable update lengths are a necessary evil that your movement code will have to be able to handle. Use a simple update smoothing algorithm to make the most of the problems go away.

SINGLE UPDATE FRAMES OF REFERENCE ARE A THING OF THE PAST. It's impossible to do coordinated unit movement without planning. It's impossible to do planning if you don't track past decisions and look at what's likely to happen in the future. Any generalized coordinated unit movement system needs to be able to recall past collision information and have future collision information available at all times. Remember that minor variations during the execution of a collision resolution plan can be ignored.

No Stupid Swordsmen

Simple unit movement is, well, simple. Good, coordinated unit movement is something that we should be working on in order to raise our games to the next level and make our players a lot happier in the process. With these articles, we've laid the foundation for a coordinated unit movement system by talking about topics such as planning across multiple updates and using a set of collision resolution rules that can handle any two-unit collision. Don't settle for stupid swordsman movement again! ■

FOR FURTHER INFO

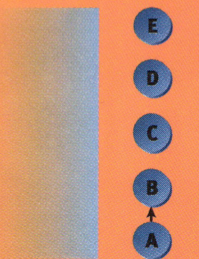
- Archer Jones. *The Art of War in the Western World*. Oxford University Press, 1987. ISBN 0-252-01380-8. This is a great book if you're looking for information on historical formation usage.

- Bjorn Reese's page of pathfinding/navigation links is at

<http://www.imada.ou.dk/~breese/navigation.html>

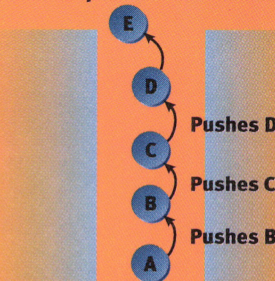
FIGURE 14. Solving the Stacked Canyon Problem.

A. Classic Stacked Canyon Problem



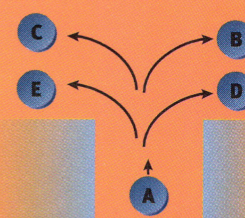
Unit A is trying to walk through a Stacked Canyon.

B. Each unit pushes the unit next in line.

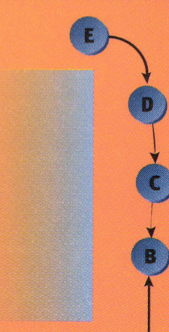


Unit A is trying to walk through a Stacked Canyon.

C. When each unit can move to the side, it does so.



D. Reverse order to rebuild situation.



Replace this unit first.

Take it like a *Game Developer*

**If you're a
reader of
Game Developer
magazine, we've
got a soapbox
reserved in your
name.**

Point your browser to:

<http://www.gdmag.com/subserv/>
and take the 1999 *Game Developer*
magazine subscriber study.

Sound off about the magazine,
the industry, and what you do for a
living, and get a chance to win 1 of
2 \$250 American Express Gift
Certificates.*

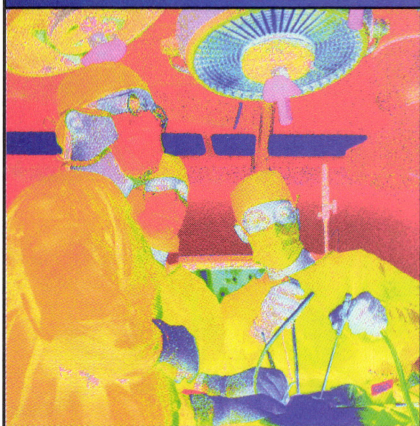
This opportunity to be heard
(confidentially, of course), comes
around just once a year, and ends
on **January 31st, 1999** - so take
just a few minutes and get your
message across.

Take it. You know you want to.

<http://www.gdmag.com/subserv>

*Only current subscribers of Game Developer magazine who fully complete the survey are eligible to win the \$250 gift certificates. Two winners will be chosen at random in February, 1999, and notified by email. You will need your nine-digit customer number from the mailing label of the magazine to take the survey.

ON THE FRONT LINE OF GAME INNOVATION
**Game
DEVELOPER**



Shiny Entertainment's WILD 9

by Didier Malenfant

S

hiny Entertainment is well known for its successful console platform games. And yet, huge hits such as EARTHWORM JIM and its sequel have resulted in huge expectations for future projects within this genre. Game players who might forgive minor imperfections in an innovative

PC title, such as Shiny's MDK, are less understanding of faults in a platform game. The Golden Age of platform games, ushered in by the NES,



the Super NES, and the Megadrives consoles, established some very high standards for this genre. In this environment, our team at Shiny launched the development of WILD 9.

Didier Malenfant is a programmer at Shiny Entertainment. Even being French he is only mildly rude and enjoys snowboarding, driving his yellow car, researching in all aspects of 3D programming and running linux boxes. He can be reached at dids@shiny.com.

Like None Before

We were fortunate in that Tom Tanaka, *WILD 9*'s lead designer, had been one of *EARTHWORM JIM*'s designers. His experience and love for the platform genre guided the project's efforts from the beginning. During some of the early design meetings, we decided that one way to differentiate our platform game would be to come up with creative ways of getting rid of the enemies — most platform games involve some form of the player jumping onto the enemies' heads. So from the very start, we designed *WILD 9* to be different in this respect. From this beginning point, we realized that the main character would have to possess some kind of weapon that could eliminate the baddies in a lot of different ways.

David Perry, Shiny's president, took an interest in our initial design and challenged us to create a weapon like none that had been attempted before in a videogame. Our first ideas centered on a female character wielding a glove that could remotely vaporize enemies or objects. This glove would deliver the same kind of actions you could perform with your own hands, only with a thousand times more strength. And then, somewhere in the early design stages, we realized that this wasn't a weapon at all. The glove evolved into a beam that came out of the back of the main character's suit. This system was to take center stage in the game's design and eventually was dubbed "The Rig." *WILD 9* was to become one of the most violent games around, but ironically the main character was to have no weapon.

Kevin Munroe, character and story designer and lead animator on the project, expanded Tom's first attempts and created a whole universe and storyline for the game. The main character became Wex Major, a male teenager lost in another galaxy. Wex's encounter with The Rig gives him the opportunity to fight the evil creature controlling the planet upon which he's just landed. As Kevin described his own design, "Imagine if George Lucas co-wrote *Star Wars* with Lewis Carroll. And imagine if George Lucas then codirected it with Tex Avery." Wex soon finds allies (eight of them), and together the *Wild 9* embark upon a "David vs. Goliath" battle against Karn, a 376-year-old being with the power of a god and the temperament of a toddler. Karn has set his sights on harnessing the ultimate power of The Glove and Rig, as well as the only being capable of using them: Wex Major. From his humble anti-hero beginnings as a pizza boy in earlier designs, Wex was now the planet's only hope, and the task ahead of him was nothing short of incredible.

Because Shiny had had luck with licensing its properties in the past, we created a game bible that contained the character profiles, as well as multiple sketches of all the characters in the game. This document was then used to show production studios and other interested parties the game universe from which TV shows or toy lines could eventually draw. As David had the chance to say in multiple speeches he made on the subject, a game bible is invaluable in the quest to license your game worlds and characters. In our case, it was also a useful reference to the huge database from which we were going to create a game. Kevin and Tom designed a lot of game content — to the point where we nearly had to lock them in Shiny's basement to stop them from adding anything else. In addition to the main character Wex, the designers came up with a sidekick named



The WILD 9 team — Standing up from left to right: Erik Drageset, Jean-Michel Ringuet, Scott Herrington, Malachy Duffin, Tom Tanaka, Gavin James. Sitting down from left to right: Lori Perkins, Rich Neves, Kevin Munroe. Not pictured: Klaus Lynged, Stuart Roch, Lloyd Murphy, Didier Malenfant. And the Big Grub guys: John Alvarado, Brandon Humphreys, Mike Winfield, Ron Nakada, Neil Hong.

B'Angus; Nitro, who is allergic to everything and has a bad tendency to explode when he sneezes; Henry, who helps navigate the water levels (Wex doesn't know how to swim); Crystal and Boomer, the game's female characters; the multiple bad guys grunts Wex encounters along the way; Karn, the evil giant who faces Wex at the end of the game; the famous Little Evil Green Men (or LEGM as we later called the tiny, single-eyed green pests); Filbert the sniper; and many other characters, as well as the world they inhabit.

The initial game bible only served as *WILD 9*'s starting point, because we added a number of elements to the design as we went along (we even managed to finish some of them). Still, this document proved to be such a valuable resource that all of Shiny's future titles will start with the development of a game bible. Many times, titles are late because certain gaps in the original design were overlooked and the full design was never really laid down on paper before development started. Writing down design details forces the designers to make all the micro-decisions that the design encompasses and leaves less room for interpretation or hesitations from the rest of the team.

WILD 9

Shiny Entertainment

Laguna Beach, Calif.

(949) 494-0772

<http://www.shiny.com> or <http://www.wild9.com>

Team Size: Thirteen full-time developers at Shiny. Five contractors from Big Grub.

Release date: October 1998.

Intended platform: Sony PlayStation

Hardware used: Loads of very powerful PCs and PSY-Q devkits.

Software used: 3D Studio Max, Photoshop, Painter, PSY-Q development system, Slick Edit, Codewright, Visual C++, Sourcesafe, and custom tools.



B'Angus

WILD 9's development began around the time that the first all-around 3D platform games had started to appear. Faced with this emerging trend, we had to make a choice whether to follow it or to stick to more traditional 2D-based game play, even if it meant evolving in a 3D environment. The second solution was chosen mainly because of The Rig. We wanted full freedom for the beam to move, and that meant using the full range of joystick inputs just to achieve that goal, never mind running around in 3D. So we decided that the main character would travel on a pre-defined path that would roam through a 3D universe, involving changes of direction and multiple camera angles. This decision also allowed the level designers to carefully place camera angles along the character's path, thus enabling dramatic scenes and varied views of the scenery. Later, we used the camera to give players more awareness of their surroundings and warn them



of the dangers ahead by panning the camera one way or another.

While WILD 9's design was challenging from the start, we also wanted the game to push new limits on the technical front. We wanted to deliver not only great and innovative game play, but also amazing visuals. The results were a mixed bag, and our technological effort was probably the most tumultuous part of the game's development.

The Tools and the Talent

WILD 9 was originally targeted at two platforms; the Sony PlayStation and the Sega Saturn. We eventually dropped the Saturn version, much to the disappointment of many Sega fans worldwide. Still, our original plan influenced the technical side of the project throughout the course of the game's development.

Characters and levels in the game were modeled in 3D Studio Max and then exported using a proprietary plug-in that Malachy Duffin wrote. In order to account for both platforms, Malachy first wrote the export plug-in to handle both output formats, and the scripts that drove the export process contained lines referring to both the PlayStation and the Saturn. When we dropped the Saturn version, half of the data in these scripts became obsolete; from that point on, only the PlayStation side continued to evolve.

The export plug-in handles the character's geometry, which is stored as a certain number of limbs, and animation, which is stored as translation and rotations of those limbs. This system has a small memory footprint while remaining flexible enough for our purposes. (We didn't use single skin meshes in the game because the on-screen size of our characters would have rendered this refinement invisible.) One of the game engine's features was the ability to hide and unhide specific limbs, which was pretty handy in optimizing the game's rendering time. One of the tricks we used was to produce multiple level of detail (LOD) models of certain objects, and store those LODs as separate limbs within the same character file. We could then hide and unhide LOD limbs depending on an object's distance from the viewpoint; once again, the small size of our objects on



the screen made the popping from one LOD to another minimal.

The levels and characters also contain various invisible bits of information, such as trigger points, reference points, and collision boxes. A level's trigger points generate monsters and powers ups. A model's reference points provide anchors for multiple effects. If a character, such as the alien beast in the Beast Engine level, has smoke coming out of its nostrils, then these will be generated from a reference point placed on the character model. The reference point can be hidden and shown according to a visibility track placed in the model's animation in 3D Studio Max, which makes editing them simple. Of course, things are never too simple. We encountered some problems toward the end of the project when time came to get the PAL version of WILD 9 going. The game's final level, which features an encounter between Wex and Karn, involves Karn trying to catch up to a running Wex. Because Karn is such an imposing fellow, his



Filbert

footsteps generate a puff of smoke and a loud bang; we generated these footsteps with reference points placed on Karn's feet. A problem arose with the PAL version because the animations played at a different speed to compensate for the 50Hz frame rate. Some of the reference points were skipped altogether, and Karn would appear to limp as only one leg would make a sound. We solved this problem by making the reference points active for multiple frames and preventing the game engine from processing those two frames in a row. If one reference point was skipped, we were sure to catch at least one another from the same group. It wasn't a very elegant solution, but it served its purpose late in the project.

All in all, 3D Studio Max was a decent editing environment for Stuart Roch, Lori Perkins, and Rich Neves to work with, although a dedicated editor would probably have been better in some cases. 3D Studio Max offered near-total layout freedom, but was lacking in the ease-of-use department, considering that we weren't even using a tenth of its features for our level-building and real-time character purposes. One feature that was very useful to us, however, was the user-defined properties it allows you to input for each object. By including a lot of keywords and properties within each game object, we could easily pass on the information that the export plug-in needed to and process its output accurately and efficiently.

My assignment for the project was to program the game's intermediate levels, which consisted of jetbike races chasing bad guys in various landscapes, falling down a huge tube trying to avoid projectiles, and the game's final encounter between Wex and Karn.

The tools and the game engine also allowed for animated textures and particles, which the team's artists used extensively. Jean-Michel Ringuet used animated textures in the Crystal Maze level, for example, to give crystals a extremely realistic glow. Erik Drageset provided levels such as Drench, which featured a series of multiple water falls and water effects — it had us all in awe the first time we saw it. In the meantime, Lloyd Murphy was working on Wreckage with the secret intention of making it the most graphically intensive level in the whole game. He kept that record to the end of the project

with only a few kilobytes of memory and a few bytes of video memory left when this level was running. However, toward the end of the project, when we'd add a game element that would appear on all the levels (such as the in-game information panel), Wreckage would be the first level to break and run out of memory. We used the particle system extensively throughout the game for explosions and various pyrotechnics. Here too, unexpected usage of the particle system proved very effective in the Drench level, where raindrops fall and create very

realistic ripples into the water — all done using different types of particles.

Two factors led to the literal flourishing of particles everywhere in the game. Gregg Tavares, who started as a programmer but left the project midway through development, added the ability to create and use particles from inside the game's scripting language. Furthermore, John Alvarado built his now famous Particle Studio module. This module allowed us to preview the effects of the particle system's many parameters while modifying them in real-time using the joypad.

Power that gives peace of mind

from **premia**
Premium Quality Software

Codewright
Programmer's Editor

With this kind of power,
you won't have to worry
about what's around
the next corner.

IDE Integration **Web Leverage** **Y2K Flexibility**

Premia Corporation
9615 SW Allen Boulevard
Beaverton, Oregon 97005
Phone: +1.503.641.6000
Fax: +1.503.641.6001
sales@premia.com

Download the trial version now:
<http://www.premia.com>

Codewright and Premia are registered trademarks of Premia Corp.



These kinds of tools, together with our scripting language, offered our artists a creative freedom that paid off for us in a big way; it's likely to serve as a model for our future development efforts. Great and unexpected things resulted when the artists were able to try out new ideas out without involving the programmers. A typical scene involved a bunch of programmers looking at the artist's screen and saying something along the lines of, "I didn't know it could even do that."

On the other side, every time a task required the programmers' help, synchronization became a major problem, with the programmers trying to fit these extra tasks into an already busy schedule and artists waiting around for a very long time before they could wrap up their levels. One perfect example of this was the game's enemy AI. Even though the artists could place enemy actors anywhere in their levels, they always needed a programmer to code at least a place holder AI routine so that the artists could preview their work. As a solution, we built a test level that we could use as an object viewer. Artists could drop their models into this test level to get a preview of what they would look like in their final, real-time, PlayStation form. While the test level helped, it didn't solve completely solve our work synchronization problem. Enemy behavior still needed to be coded in for the level to be fully tested and polished.

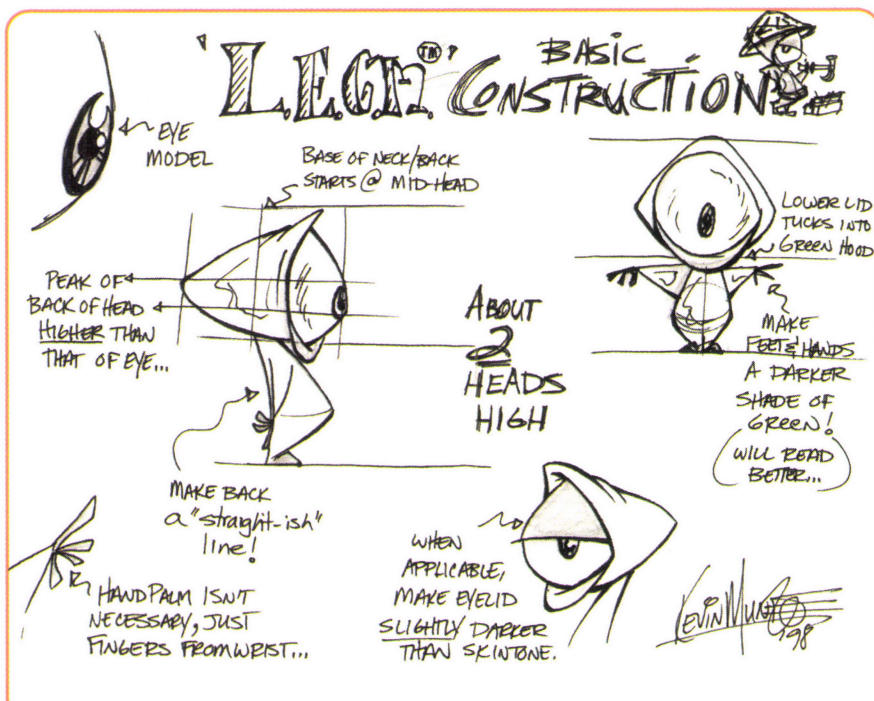
Still, a really effective solution to many of the problems that we experienced involved not only extra organization and proper scheduling, but also better communication. If a programmer is unaware that another team member is waiting on a certain feature, then the process would end up in a gridlock situation. So, while the organization we chose worked, it was never perfect. We used Microsoft SourceSafe to share code between the programmers, but the only

trusted version of the game was always living on Gavin James's computer. Gavin personally made sure that he'd incorporated every change we made to the game by keeping track of everybody's progress and updates. For future projects, we hope to implement some kind of registry or database where all the game elements can be recorded. Our hope is that such a system would allow us to better see how changing some part affects all of the other parts.

Our programmers developed a scripting language to enable easy implementation of the characters' AI. The language is based on a byte code, which is interpreted in real-time during the game and specifies three main threads for each character. The intelligence, animation, and movement threads control the actor's behavior and also have the ability to spawn additional threads, if necessary. The scripting language also handles collision events using a table that

contains all the different classes of characters in the game and the corresponding collision types allowed to occur between them. The system can thus trivially reject collision tests between objects that should never collide with each other, such as two static objects.

As Gavin continued to improve the game engine, we kept extending the scripting, adding new features and allowing access to just about any low-level function through C function calls. Though imperfect, the scripting language was fairly easy to use once all its loopholes were known, and it enabled faster prototyping than straight C. A testament to the versatility of the scripting language is the fact that even the menu system that Malachy wrote is, in fact, treated as a normal game level by the engine. Every menu item that the players see on the screen is a game actor and the background is a huge game level.



The Little Evil Green Men



Progress

WILD 9's development was divided into two relatively distinct periods (more on the reasons for this later). The first team that worked on WILD 9 had most of the game engine written, a few levels started, and some game play mechanics implemented, but the game itself was not what we could call playable. When the second team took over, Gavin spend most of his time cleaning up the engine, while the artists and level designers focused on finishing one level. This was the long birthing of Gulag, which took a full two months to finish; more than any other level in the game.

To the team's credit, a second level, called Bombopolis, was started during the last few weeks of Gulag. Then things started to really snowball. People were getting used to the tools and the limitations of the engine, and we spent the remainder of the project making progress at an extremely scary pace. At this point, the engine was getting pretty stable, and we were able to start adding new features. Levels started popping out of nowhere, and before we knew it, we were playing a full-blown game.

What Went Right

1. THE SECOND TEAM. Shiny had originally wanted to release WILD 9 last year. Back then, another team was working on the project. But following a major staff upheaval, the only people left from the original staff were Tom and Kevin; the rest of the first WILD 9 team, in one way or another, left Shiny. I'll elaborate on this in the "What Went Wrong" section, but needless to say, inheriting a project, engine, graphical work, and early levels from another team is indeed not an easy task. A constant tension exists between trying to work with old stuff and wanting to redo it all over again.

Most credits must go to Gavin, who had to abide by technical decisions made by other programmers before him; he spent a lot of time trying to make sense of the game engine's features and inner workings. Word is that even though the game is now finished, he's still somewhere back there trying to make sense of part of it.

In truth, he spent a lot of time trying

to anticipate what people would need in order to be productive and gave, where possible, top priority to other people's needs to avoid gridlock and keep things moving. He also helped a lot by sorting out small and easily definable tasks and handing them to people who were learning parts of the system, thus making their learning curve easier.

If the first team's working atmosphere was not perfect, that of the second team was a completely different story. People were gradually added to the team as development progressed, and the last few months were spent

under incredible pressure and extremely tight deadlines. And yet through all this, we get along well with one another, and no homicides took place.

Joking aside, the chemistry among the second team was incredible, and I think I can safely say that everyone had a great time working on this title. A great atmosphere meant suggestions on each other's work were well-taken, spirits were up, and productivity was positively influenced.

2. THE SCRIPTING LANGUAGE. As I mentioned earlier, we used the scripting language for just about everything

hey- game artists!

You know you make some of the coolest digital art out there. So give your reels the attention they deserve. Submit to the Game Developers Conference 1999 Game Art Screening Room.

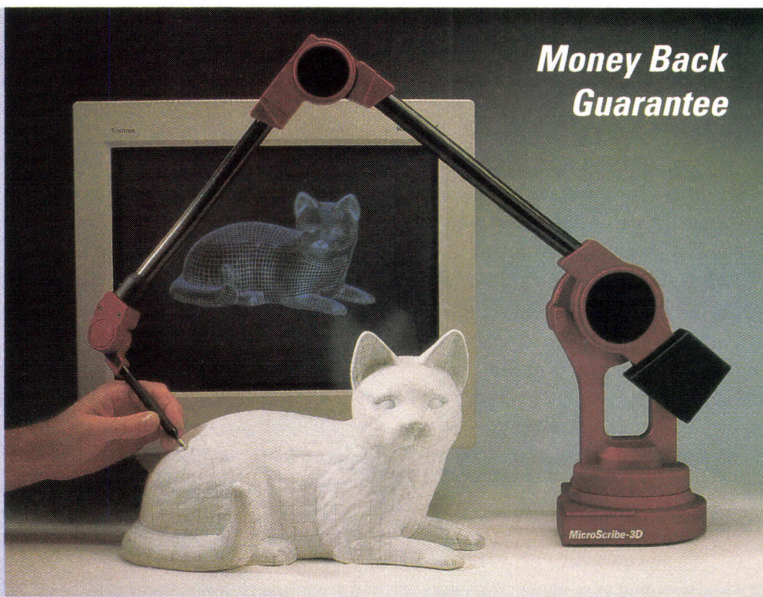
Here's your chance to have your best game animation shown off big time to your colleagues and potential employers. Only the most exceptional clips will be plucked from the masses, judged by an independent panel, and honored in a continuous showing at the GDC!

The deadline is January 29, 1999.

**gameart
screening room**

For submission guidelines: www.gdconf.com/screeningroom

MicroScribe-3D™



**Money Back
Guarantee**

The only 3D digitizing system with a refined look, feel, and performance. The MicroScribe system allows rapid 3D digitizing with 0.009" accuracy over a 50" spherical workspace. Quickly create impressive 3D models for use with 3D Studio™, Lightwave™, Alias™, SoftImage™, Strata™ and other popular 3D packages. The desktop system has been ergonomically designed using light-weight graphite and precision counterbalancing to allow effortless manipulation. The award winning MicroScribe-3D is robust, attractive, and easy to use. It is simply the *fastest* way to create 3D models for animation, engineering, multimedia, or virtual reality.



Immersion Corporation New Drivers for SGI
(408) 467-1900 Fax (408) 467-1901
(800) 893-1160 ems: info@immerse.com

Visa, Mastercard, and
American Express accepted

in the game. In fact, the language turned out to be rather easy to learn, and it was very powerful once some key features were added to it. It was a great way to prototype certain actors very quickly and work with the animator and the level designer to polish up behaviors and attack patterns. The interpretation of the byte code wasn't the fastest thing around, but it turned out to be a positive trade-off for flexibility and ease of use. Also, most of the low-level functions were performed by straight C code that was triggered by scripting commands from within the actors' AI code.

3. STUART ROCH STEPPING UP AS A PRODUCER. During a rough patch in the second team's initiation, when we thought things couldn't get much worse, WILD 9 lost Scott Herrington (it's producer), who had to replace the departing Simon Cox as Shiny's PR manager. Stuart Roch had been hired initially as an assistant designer, but he quickly stepped up to full-blown producer, even though he only officially earned the title later on in the project. Stuart's attention truly helped keep the project on track, as he did an awesome job at keeping everything together, organizing schedules, and making sure that we as the team had everything we needed. It's no surprise that now, after WILD 9, Stuart will go on to produce other Shiny titles.

4. THE ADDITION OF BIG GRUB. Toward the end of the project, it became clear that we weren't going to finish all the ideas that we had had for this game if we didn't get serious help in the form of extra manpower. We needed art for the menu system, a closing movie for the end of the game, and a dozen programmers to implement the behavior of all the actors that the level designers had in mind for their levels. We didn't get a dozen programmers, but we did decide to flout Brook's Law and add more people to the project. Learning from past mistakes, we looked for people who were not only talented, but work well with the rest of the team. Nobody fit this bill better than the Big Grub guys.

Big Grub is a small company from Irvine, Calif., whose talents we contracted to help us out with all the content ideas we had for the game. John Alvarado, Ron Nakada, and Mike Winfield helped program the AI for

**Changing The Way
The World Develops
Real-Time 3D
Graphics**

www.viskit.com
Free Vampire Demo Available!

A New Real-Time 3D API For Windows

ADVERTISER INDEX

NAME	PAGE	NAME	PAGE
3Dfx Interactive	25	Macrovision	15
ATI Technologies	18	Metrowerks Inc.	29
Cakewalk	8	Multigen	31
Caligari Corp.	4	Numerical Design	11
Conitec Datensysteme	68	Okino Computer Graphics	14
Creative Labs	2-3	Paradigm Entertainment	66
Dell Computer	23	Premia Corp.	63
Evans & Sutherland	7	RAD Game Tools Inc	C4
Globix	14	Sierra Studios	26
IBM	C3	Silicon Graphics	C2-1
Immersion Corp.	66	Square USA	70
Incredible Technologies	69	Stainless Steel Studios	71
Intel	21	Stormfront Studios	69

Miller Freeman Game Group Research

3D GRAPHICS MARKET OUTLOOK

IF THE 3D GRAPHICS MARKET IMPACTS YOUR BUSINESS, THEN YOU NEED EXPERT ANALYSIS. Omid Rahmat, Principal, Doodah Marketing, and industry consultant, has taken an in-depth look at the hottest technologies and the companies creating them. Who owns the market? Who is up and coming? Where should YOU place your development resources in 1999 and beyond?

This 30 page report will help you target and support the right hardware while cutting down your production schedule. Help avoid costly mistakes by taking advantage of this detailed report, including:

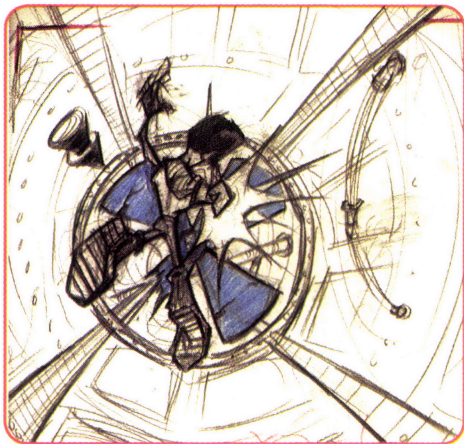
- The factors driving consumer 3D
- Projected sales through 1999
- Movers, shakers, influencers and up-and-comers
- The impact of the 3D API
- OEMs role in Consumer 3D
- Company information from 3Dfx, 3Dlabs, Cirrus Logic, S3, nVidia, ATI, Evans & Sutherland, Intel, Intergraph, Fujitsu and more!

The 1999 3D Graphics Market Outlook is the only report designed specifically for the game development community, giving you exactly what you need to make great 3D games, on time and on budget.

To find out more, or to pre-order your copy at the special introductory price of \$495, please see: www.mfgame.com/research/

AVAILABLE Q1 1999

Miller Freeman Game Group



68

many of the game's characters, Brandon Humphreys worked on the interface art and the end movie, while Neil Hong designed the second falling level and helped with the end movie. We couldn't have finished the game in time without them, but we also took a big risk bringing on new people toward the end of the project.

5. KEEPING EVERYBODY IN CLOSE QUARTERS.

Shiny is housed in a three-floor building. The first floor is mainly administrative, the third floor houses the MESSIAH and SACRIFICE teams, which leaves the second floor for WILD 9 and STUNT RC COPTER. Shiny has no private offices, so to speak; both development floors are organized in an open space layout, with separating half-walls in some cases. STUNT RC COPTER is only a three-person team, so the rest of the space was occupied by the WILD 9 team. While we weren't cramped, it's safe to say that some people got to know their neighboring coworkers rather well.

This proximity became invaluable because it enhanced communications

among us ten-fold. One early problem with the project involved assigning tasks to the individual team members and keeping abreast of who was up to what. Weekly meetings going over everybody's schedules and a lot of direct and constant communication among us (most of the time without even leaving our desks) directly addressed this problem.

What Went Wrong

1. THE FIRST TEAM. I once heard David Perry say that the first WILD 9 team was a great collection of talented individuals, while the second was a talented team. This statement alone sums up the main problem encountered by the first team. As we found out later in the project, communication and good atmosphere were two factors that made this project move forward; it's now obvious that the best thing to do was to start over rather than try to salvage a situation that was going nowhere.

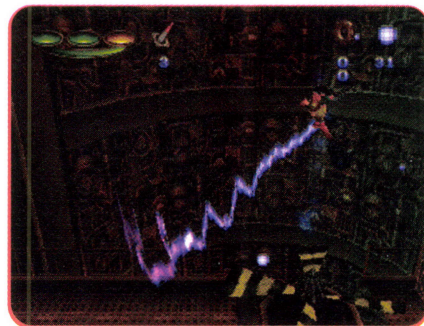
2. A LACK OF EXISTING DOCUMENTATION. I've already sung the praises of our scripting language, but not everything was totally perfect in this picture.

Although it ended up being very useful, it became clear early on that the language and its op-codes had massive loopholes and "gotchas" waiting to jump on us. To help other programmers with their first steps, Gregg started maintaining a document as he was learning the AI system. This document described all the major problems that he'd encountered and ways around them that he'd discovered.

When Gregg left the project, Gavin and the rest of the programmers took over the responsibility of updating this document as new problems appeared or new solutions were found.

This process continued until the very end of the project. The document was invaluable in getting somebody up to speed on the system, such as when the new programmers from Big Grub came to the rescue. Overall, the language's design was, in some places, questionable; for instance, the actor registers were called params and the virtual processor's variables were called regs, which was greatly confusing.

3. THE PHYSICS ENGINE. The physics engine turned out to be one of Gavin's biggest nightmares during the WILD 9's development. He inherited



Create
your own
cool 3-D Games!



3D GameStudio

Authoring System
for 3-D Realtime Computer Games

- ◆ Create cool 3-D adventures **without programming!**
- ◆ Contains World Editor and the **ACKNEX 3-D engine**
- ◆ Free 3-D template game with 150+ textures included
- ◆ 3-D polygonal landscapes with slopes, bridges, buildings
- ◆ Player can walk, drive, fly, climb, swim, dive...
- ◆ Self-defined panels, overlays, cockpits, and scrolling text
- ◆ Create your own objects, actors, walls, weapons, menus
- ◆ 320x400 256 color smooth scrolling VGA resolution
- ◆ 8-channel stereo sound & midi support
- ◆ Imports PCX, LBM, MDL, WAV, MID and IBK files
- ◆ 200+ pages English manual with game tutorial

3D GameStudio Lite \$99

3D GameStudio Commercial \$199
(+ SVGA + 2-Player mode)

3D GameStudio Professional ... \$1450
(+ Polygonal Actors + CD-Audio + FLIC player)

Prices + \$20 for overseas shipping / \$40 for additional email delivery
Infos, demos, ordering → <http://www.conitec.com>

CONITEC

Germany ■ D-64807 Dieburg ■ Dieselstr. 11c
Tel +49 6071 92520 Fax 925233 ■ www.conitec.com

this code from the previous programmers, and he spent more time trying to unravel, optimize, and just get that part of the code functioning than on any other identifiable part of the game. His advice to engine inheritors: unless the physics are perfect and written exactly as you want them, grow your own.

4. DEAL ME IN. Our tools programmer and resident Irish person Malachy once decided to bring us a little multiplayer game he was working on in his spare time. The game was called DEAL ME IN, and he wanted us to help him beta test it. DEAL ME IN is best described as a very addictive cross between Scrabble and Poker. Actually, "addictive" doesn't even begin to describe the darn thing.

Everyone at Shiny plays games. So when it comes to testing a very addictive game, we are very thorough. The game became an obsession, to the point where it was affecting our productivity. It turns out, after extensive research, that there is no such thing as a "quick game of Deal Me In."

5. THINGS THAT DID NOT MAKE IT. Tom and Kevin are two very creative

fellows. The material they came up with is enough to fill up three or four games like this one. In addition, the whole team took this game to heart and ended up coming up with suggestions of their own. We were thus faced with a situation toward the end of the project: we could not fit everything into the game.

Wex's weapon, The Rig, is normally blue. But it possesses a red mode, which is greatly under-used in the game. The red mode was initially meant to be a more powerful blue beam, but this created game play issues, as a player will sometimes want to transport enemies in the beam from one point of the level to another. The red beam would vaporize just about everything on contact, which was obviously a problem. This is one of the only design issues that was never truly fixed and the red beam, even though it made it into the game, isn't used to its full potential.

Many other great actor ideas never made it into WILD 9: for instance, the 3D cow that was intended as a reference to EARTHWORM JIM, or the tank that Lloyd Murphy modeled that was

supposed to populate the Wreckage level. In fact, Lloyd's tank became sort of a myth, which we would bring up at various meetings: "So we don't have time to fit even a tiny little tank then? Are you sure? What if I pretend it's a background?" The need to stop development at some point made it impossible to use it all the ideas and content that we had, although I hear from reliable sources that the cow did eventually make it in there somewhere.

We Did Our Best

It's an amazing feeling after working so hard on a game to sit back and start playing it. In our case, the general feeling was that WILD 9 turned out amazingly well, especially in light of its rocky development history. One of the best feelings was to see the press reaction at E3 this year, and to hear the comments that WILD 9 had come a long way. If you ever saw anything from WILD 9 a year ago, go and check out the final version. We did our best to make sure you will have a pleasant surprise. ■

69



3D Programming Opportunities for Top Talent Console & PC Computer Games and Next Gen Platforms

Racing (PSX, N64 & PC)—Stormfront's NASCAR '99 published by EA Sports is one of the top franchises in the industry. Now, we're working on the next generation and want to take this platinum hit even farther.

Baseball (Consoles and PC)—High energy baseball game with tons of action and fast, motion-captured anims. Hot technology, exciting game play and graphic flair. Major publisher with major marketing support for the game.

Fantasy Role-Playing (PC)—A top license and a top publisher, featuring key innovations in game play. We're building a new engine from scratch.

Action (PSX, N64 & PC)—We're building a new 3D action game featuring wild 3D action as vehicles careen through loop-the-loops, corkscrews and jumps against a background of fantastic landscapes.

We're adding to our award-winning development staff of 70 and looking for a Technical Director, Sr. and Lead PSX and N64 Programmers and Senior and Lead PC Programmers. You must be able to prove that you can play an integral role in creating the next generation of blockbuster game titles.

Please send your resume to:

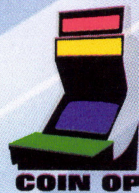
Stormfront Studios, Attn: Marta Daglow, 4040 Civic Center Drive
San Rafael, CA 94903 Fax: 415-461-3865, e-mail: mdaglow@aol.com
www.stormfrontstudios.com

INCREDIBLE TECHNOLOGIES IS GROWING!

We are a dynamic company with a 15 year track record of success in the game industry. We offer a modern, pleasant office environment, competitive salaries and an outstanding benefits package.



PlayStation



COIN OP



NINTENDO 64

WE are looking for...

- PSX Senior Level Programmer
- 3D Senior Level Programmer
- Senior Systems Level Programmer

Are YOU...

- Hard Working
- Self-Motivated
- Experienced
- A Serious Gamer

Contact us at:

jsieglist@itsgames.com
www.itsgames.com

Incredible Technologies, Inc.
1600 Hicks Road
Rolling Meadows, IL 60008
attn: Jane Sieglist

Nintendo, the official seal, 64 and the 3-D "N" logo are trademarks of Nintendo of America, Inc. © 1997 Nintendo of America, Inc. All rights reserved. PlayStation and the PlayStation logo are trademarks of Sony Computer Entertainment, Inc. All rights reserved.

They called you nimrod,
dweeb, propeller head,
and geek.



NOW
it's time to
kick some butt.

Kick your career into high gear at SQUARE USA. It's an opportunity that can't be beat.

We are looking for people with game development experience for the following positions:

Game Programmer
Assistant Game Designer
2D/3D Artist

Apply to:

SQUARE USA, Inc.

4640 Admiralty Way, Suite 1200 Marina del Rey, CA 90292

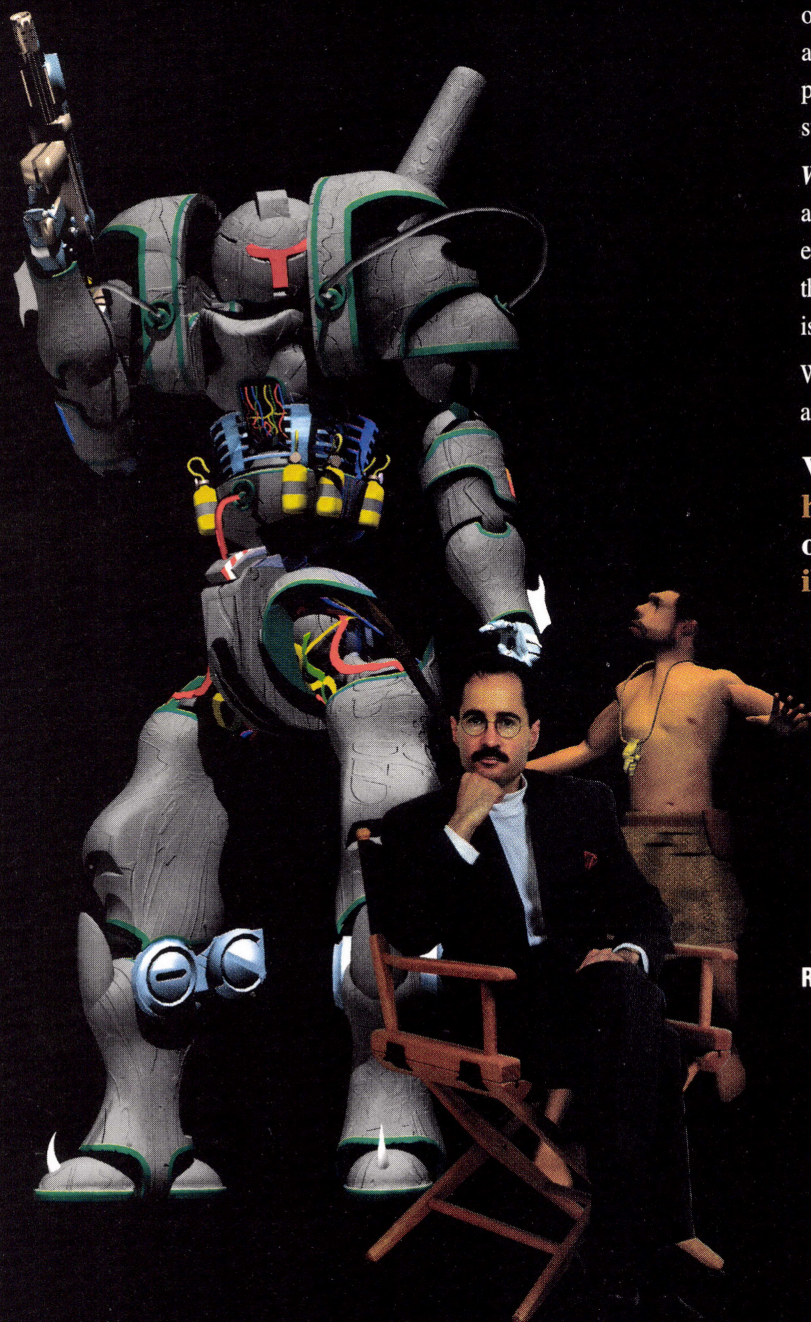
Fax: (310) 302-9550 Email: hi@sqa.com

No phone inquiries please.

SQUARE USA



Join The Team That's Developing Premier Real-Time Strategy Games For The Next Millennium



Stainless Steel Studios, Inc., founded by Rick Goodman (creator/lead designer for Microsoft's Age of Empires®), and based in Cambridge, Massachusetts has embarked on the development of real-time, multi-player strategy games for the next millennium.

Here's a unique opportunity to participate in the creation of innovative, top-quality game titles and become part of a new, fast-growing game development company that is poised to become the industry leader in the real-time strategy game genre.

We believe that success is based on teamwork. If you are a die-hard gamer who is hard working, self-motivated, energetic and you are interested in having major input into the creation of 'AAA' game titles, Stainless Steel Studios is looking for you!

We're looking for the very best artists, character animators and senior game programmers in the industry.

Visit our web site:

<http://www.stainlesssteelstudios.com>

or contact us at:

ilovemyjob@stainlesssteelstudios.com.

All inquiries will be kept strictly confidential. We are an equal opportunity employer offering an excellent salary and benefits package.

Rick Goodman: *Recipient of the 1998 Computer Game Developer Conference's "Annual Achievement Award for Game Design and Development".*



Garage Developers are Not Dead

Let me start by clarifying some terms, particularly "Garage Developer" and "Suit." Do not confuse a "Garage Developer" with a "Geek." Likewise, do not mistake a good

"Businessman" for "Suit." These terms are about opposing mindsets.

Those of the Garage Developer mindset, given any limits, would rather work their asses off and achieve something than do nothing and achieve nothing. Garage Developer, as I use the term, describes the innovative, self-starting, and hard-working mindset of the individual who has a strong inner vision and follows it. A Geek lacks these qualities.

The Suit mindset, as I use the term, would rather rip-off an idea than innovate. He is not like the honorable Businessman, as his goal is to crank out crap (such as all the Myst rip-offs), dump it in a box, and sell it. This mindset believes that pulling ads is saving money, and that lying on the box will promote sales. This mindset has a short-term view of things.

What we have in the industry now is a great number of Geeks and Suits creating rip-off garbage, screwing each other over, and filling the bargain bins to critical capacity. What we need are Garage Developers and Businessmen working together to create innovative and profitable games.

So, I am sick to death of the statement "Garage Developers are dead." I think the statement was originally intended to mean that the days of one guy developing an entire game by himself are over (and I don't even agree with that). You'd be surprised at the number of two- and three-person shops out there that are responsible for huge chunks of the development process on some major titles. And it's not just programmers. An informal survey of mine

revealed that many (who wish to remain anonymous for fear of appearing "unprofessional") are active in the production of 3D models, animation, music, and even design. The small guys are always rising to the top. It's a cycle; the small guy succeeds, becomes a big guy, and another small guy pops up to do things even better.

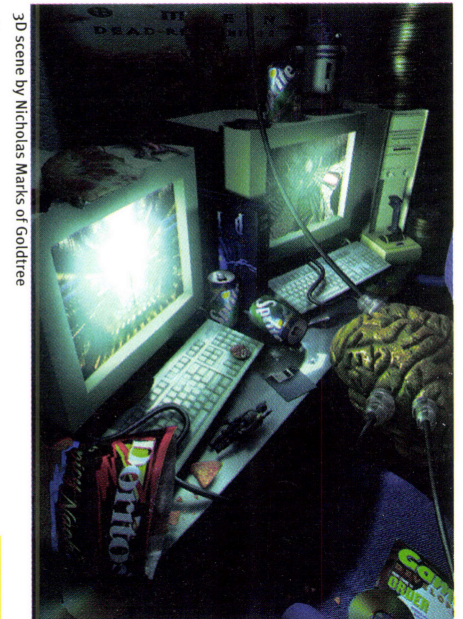
People will latch on to a statement such as "Garage Developers are dead" and misconstrue it for their own purposes — usually to justify their own failings. People have been echoing this sentiment about garage developers and other innovators for hundreds of years. The American patent office declared that everything that could be invented, had been, in the late 1800s. Obviously, a few new inventions have appeared on the horizon since that time.

This statement reflects more than just a pessimistic view of our industry, but a core problem of humanity. It says to me that some people believe that all innovation, individual achievement, and accomplishment is over. That people believe that there will be no more daring startup businesses based solely on a grand vision and some elbow grease. That people believe that there will be no more technological breakthroughs based on imagination and perspiration. In short, that there will be no more great works by great people. Everything has been done, discovered, and dared — so why try? "So why try?" is the pure essence of the loser mindset.

So what's my advice for making it as a Garage Developer? It's easy to detail.

1. Write your game idea on paper.
2. Build a prototype.
3. Submit it to a publisher.
4. Develop the game.
5. Detail everything you did right and wrong in steps one through four and write a Postmortem for *Game Developer*.
6. Start at step one and repeat this process until you succeed.

Dare to take these steps and repeat them until you succeed. Each time around you will make progress towards producing a successful game. This goes for the professional game developers as well. Is your company or development team reinventing itself, stretching itself, and daring new heights each time you start a new project? If not, then you are moving backwards. Whether you're a true Garage Developer or not, have a mindset of quality and innovation. It's a mindset vital to the game development community. If you decide to take this daring journey, you will not be alone. You'll find plenty of help and information online, in news groups and web sites. The Internet has hooked all of our garages together. We now have tools and information at our disposal undreamed of only a few years ago. Garage Developers have never been more powerful. ■



Luke Ahearn is lead designer and producer at Goldtree. He is currently developing Goldtree's next two titles. He can be reached at luke@goldtree.com.

IT'S RENDER HAPPY. Imagine your apps with cartoon rockets strapped to their feet. That's how they run on the new IBM IntelliStation.[™] Its two screaming microprocessors make applications from vendors like Adobe, Avid, and Alias|Wavefront literally scream with delight. What's that mean to you? Less time waiting. More time creating. Oh yeah, and happy killer graphics. Visit www.ibm.com/intellistation or call **1 800 IBM 7255, ext. 5007**. Careful, though. What you find could render you speechless.

THE IBM INTELLISTATION WORKSTATION.[™] Windows NT[®] / Up to two Intel Pentium[®] II Xeon[™] processors (450 MHz) / Intel 440GX AGPset with 100 MHz front-side bus / Up to 2GB SDRAM ECC memory / Matrox Millennium II / G200 / Permedia 2A / Intergraph Intense 3D Pro 3400 / Starting at \$1865.*

 e-business tools

IBM[®]



*Estimated reseller price to end users for model 6893-10U. Actual reseller prices may vary. Monitor sold separately. MHz refers to microprocessor speed. Certain factors will affect application performance. PCs referenced in this ad include an operating system. Other company, product or service names may be trademarks or service marks of others. IBM product names are trademarks of International Business Machines Corporation. Intel Inside and Pentium are registered trademarks and Pentium II Xeon is a trademark of Intel Corporation. © 1998 IBM Corp. All rights reserved.

The Best in Game Development Technology!



Introducing Bink - the new true-color codec from the author of Smacker!

Jeff's done it again! Bink is now available and revolutionizing game videos just like Smacker did four years ago! Bink is a "better-than-MPEG-II" class codec. Yup, that's right - **better than DVD!** It produces higher-quality than MPEG II and is up to three times faster than other software decoders. Now there is finally a true-color codec good enough for game developers.

Bink is a hybrid block-transform and wavelet codec that can encode your video using 16 different compression techniques (I42 wavelet, DCT, motion compensation, a variety of vector quantizers, Smacker-style, etc). With all of these techniques in one codec, Bink can handle almost any type of video. Bink also includes a psycho-acoustic based audio codec that is capable of 8 to 1 perceptibly lossless compression, so your audio will sound as good as your video looks.

Bink is available now for Windows 95/NT (DOS and MacOS coming soon) and supports DirectDraw, DIBSections, DirectSound, and the Windows waveOut system. Bink uses the YUV colorspace, so it can use DirectDraw overlays for hardware color conversion and smooth scaling. Bink also includes a bunch of hand-optimized assembly YUV to RGB colorspace blitters, so you'll be able to access Bink's output in any RGB format you like. Bink is a high-end codec, and as such, it requires at least a Pentium/150 (Pentium/200 preferred). Bink really doesn't have a minimum CD requirement - 320x240 animations look great at 150 kps (1x CD) and 640x480's only need 450 kps (3x CD). Bink does need a reasonable video card that is capable of handling all the true-color pixels, though. The Bink SDK is very similar to Smacker's, so integration is easy, and upgrading from Smacker is easier still.

You're really going to love Bink - download the Bink Tools (or call for a demo CD) and try it on your videos today!



New MPEG Layer-3 decompression support!

Miles 5.0 includes a redistributable MPEG Layer-3 decoder (patent rights licensed from Thomson and Fraunhofer). MP3 provides perceptually lossless compression at 11 to 1! Your sound files will be **more than ten times** smaller and will sound exactly the same! Miles supports both on-the-fly decompression (for minimal memory use), or pre-playback decompression (for minimal CPU hit). Miles even supports MP3 compression of DLS instrument data for incredibly small MIDI files.

New high-level 3D sound support!

Miles 5.0 has an all new high-level 3D sound API which supports DirectSound3D, Intel's RSX, Creative's EAX, and Aureal's A3D (selectable at run-time). With Miles, you can have 3D audio in your game in minutes! More importantly, you won't be tied to a specific low-level 3D API, so you can use the technology that makes your game sound best.

All new digital sub-system! Faster mixing, filtering, and plug-in support! Replaces the DirectSound mixer!

Miles 5.0 includes a completely new digital audio subsystem. It includes an even-faster mixer (with MMX support), on-the-fly filtering, and a new plug-in system for adding run-time sound effect processors which can be called at the pre- or post-mix stage. Miles also now uses its own mixer even when running under DirectSound - if your game starts and stops multiple sound effects frequently, then this new feature can potentially double your frame rate!

Of course, Miles 5.0 provides all of the features you loved in earlier versions: complete digital mixing, interactive MIDI playback with a built-in DLS software synthesizer, hard drive or CD-ROM digital streaming, red book CD-audio support, powerful callbacks, and much more!



Smacker - the best 256-color codec on the planet!

Yes, Smacker is still available! Smacker is still the best codec for situations such as: 256 color games (of course), games targeting low-end machines (Pentium 133 and below), games that need video sprites or video transparency (which is much faster in 256 colors), cell-based (cartoon-style) videos, animated 3D texture compression, and very high-resolution videos (800x600 or 1024x768 - only Smacker is fast enough for videos this big).

The Smacker SDK is available for DOS, 16-bit Windows, Windows 95/NT, Win32s, 68K Mac, and PowerMac. The SDK's API is identical across these platforms, and Smacker files can be played without conversion on each platform.

Smacker supports DirectDraw, DIBSections, WinG, DispDIB, SVGA VESA, and GWorlds (MacOS) for graphics output. For sound output, Smacker supports the Miles Sound System, DirectSound, the Windows waveOut system, and the Sound Manager (MacOS).

Who's using RAD? Everyone! Here's a partial list of our customers:

Activision, Blizzard, Eidos, Ion Storm, Microsoft, Electronic Arts, Bullfrog, LucasArts, Access, Bungie, Raven, DreamWorks, Humongous, Kesmai, Monolith, Origin, Sony, Psygnosis, Papyrus, Accolade, Acclaim, GT Interactive, Hasbro, Maxis, Sierra, Epic, Westwood, Take 2, The Learning Company, 989 Studios, Iguana, Rage Software, Probe, Cavedog, Midway, Vipah, Creative Wonders, New World Computing, 3DO, The Neverhood, Core Design, Blue Byte, Dynamix, Microprose, Viacom, Hypnotix, Impressions, Stormfront Studios, Virgin Interactive, Krisalis, Sir-Tech, Byron Preiss, Time Warner, Software Sorcery, McGraw Hill, Looking Glass, Adrenalin, Legend, Spectrum Holobyte, Strategic Simulations, Software 2000, Bethesda, Trilobyte, Sega, Compton's New Media, Firestorm, Criterion, Interactive Magic, Sci Limited, Adeline, Rocket Science, Atari, Velocity, Sculptured Software, Capcom, MECC, Software Toolworks, and Knowledge Adventure.

... over 2,300 games - see our web site for title lists!

Bink, the Miles Sound System, and Smacker are licensed on a per-product or per-site basis with NO ROYALTIES WHATSOEVER!

425-893-4300

335 Park Place Suite G109

Kirkland, WA 98033

425-893-4300

FAX 425-893-9111

sales@radgametools.com

www.radgametools.com

